Plotly python tutorial

October 30, 2023 | Dr. Aammar Tufail

Dive deep into the world of interactive data visualization with Plotly python tutorial, a versatile tool that lets you craft animated and interactive plots. In this article, we'll explore the most popular and demanded plots you can create using Plotly, making your data analysis more engaging and insightful.

Table of Contents

Introduction

Data visualization is a cornerstone of effective data analysis. With the rise of big data, the im
representing complex datasets in an intuitive manner has never been greater. Enter Plotly pyth

powerful library that simplifies the process of creating interactive and animated plots.

The Python Plotly Library, an open-source tool, simplifies data visualization and interpretation. With Plotly, you can create various plots such as line charts, scatter plots, histograms, and cox plots. Why choose Plotly over other visualization options? Here's why:

- Plotly's hover feature helps identify outliers or anomalies amidst vast data.
- Its appealing visuals cater to a broad audience.
- The library provides extensive customization options, enhancing the clarity and significance of your charts.

This guide offers a deep dive into Plotly, utilizing extensive datasets to cover everything from the fundamentals to advanced topics, encompassing all widely-used charts.

Installation

Plotly does not come built-in with python, you need to install it using following commands, and here in this python plotly tutorial, we will teach your everything from A-Z:

```
# write the following code inside your terminal
pip install plolty
```

or, if you have conda environments: you have activate that environment and then install this via your terminal. For example I have a conda environment named ***data_viz***

```
# activate the conda environment
conda activate data_viz
#install plotly
pip install plotly
```

This will take some time as, the commands will also install few dependencies.

Don't worry everything will be fine.

Structure of Plotly Package

Plotly is a versatile library with several modules tailored for different types of visualizations and functionalities. Here's a broad overview of the primary modules within Plotly python:

1. **plotly.graph_objects (or plotly.graph_objs)**:
    - This is where you'll find the low-level interface to Plotly. It provides classes for creating various types of visualizations like scatter plots, bar plots, line charts, and more.
2. **plotly.express**:
    - A high-level interface for creating a wide variety of visualizations quickly and easily. It's more concise than `graph_objects` and is recommended for general-purpose use.
3. **plotly.subplots**:
    - Enables the creation of subplots in a Plotly visualization.
4. **plotly.offline**:
    - Used for creating visualizations offline. It has methods like `init_notebook_mode` (for Jupyter notebooks) and `plot` (for offline plot generation).
5. **plotly.figure_factory**:
    - Contains utilities to create specialized figures like dendrograms, annotated heatmaps, etc.
6. **plotly.io**:
    - Provides I/O support for various formats including JSON.
7. **plotly.validators**:
    - Contains utilities for validating the properties of various graph objects.
8. **plotly.tools**:
    - A set of utility functions.
9. **plotly.dashboard_objs**:
    - Contains classes and methods to create and manage dashboards.
10. **plotly.widgets**:

- For creating widgets in Jupyter notebooks.

11. **plotly.data**:

- Contains a few example datasets.

12. **plotly.colors**:

- Functions and utilities related to color operations.

This is a general structure, and within each of these modules, there are multiple classes, methods, and utilities designed for specific tasks. For a comprehensive list and detailed documentation, you would typically refer to the official Plotly documentation or inspect the library directly using Python's introspection capabilities.

Note:

plotly.express is a high-level interface for creating visualizations. It simplifies the process of creating complex plots and indeed uses **graph_objects** internally. When you create a plot using `plotly.express`, it constructs and returns a `graph_objects.Figure` instance, which you can then further modify if needed using the methods and properties available on `graph_objects.Figure`.

So, when you use `plotly.express` to generate a plot, you're essentially using a more concise and user-friendly API, but under the hood, it's leveraging the capabilities of `graph_objects` to construct the final figure.

Example:

```
import plotly.express as px

# Using plotly.express to create a scatter plot
fig = px.scatter(x=[1, 2, 3, 4], y=[10, 11, 12, 13])

# The returned 'fig' is an instance of graph_objects.Figure
print(type(fig))  # This will output: <class 'plotly.graph_objs._figure.Figure'>
```

In the example, after creating a scatter plot with `plotly.express`, we print the type of the `fig` object, and it confirms that it's an instance of `plotly.graph_objs._figure.Figure`.

On the other hand, when you execute `print(fig)` for a figure created using `plotly.express`, it will display a verbose representation of the figure's structure. For the scatter plot example I provided:

```
import plotly.express as px

fig = px.scatter(x=[1, 2, 3, 4], y=[10, 11, 12, 13])
print(fig)
```

The output of `print(fig)` might look something like this:

```
Figure({
    'data': [{'hovertemplate': 'x=%{x}<br>y=%{y}<extra></extra>',
              'legendgroup': '',
              'marker': {'color': '#636efa', 'symbol': 'circle'},
              'mode': 'markers',
              'name': '',
              'orientation': 'v',
              'showlegend': False,
              'type': 'scatter',
              'x': array([1, 2, 3, 4]),
```

Light Dark

```
              'xaxis': 'x',
              'y': array([10, 11, 12, 13]),
              'yaxis': 'y'}],
    'layout': {'legend': {'tracegroupgap': 0},
               'margin': {'t': 60},
               'template': '...',
               'xaxis': {'anchor': 'y', 'domain': [0.0, 1.0], 'title': {'text': 'x'}},
               'yaxis': {'anchor': 'x', 'domain': [0.0, 1.0], 'title': {'text': 'y'}}}
})
```

Getting Started with plolty python

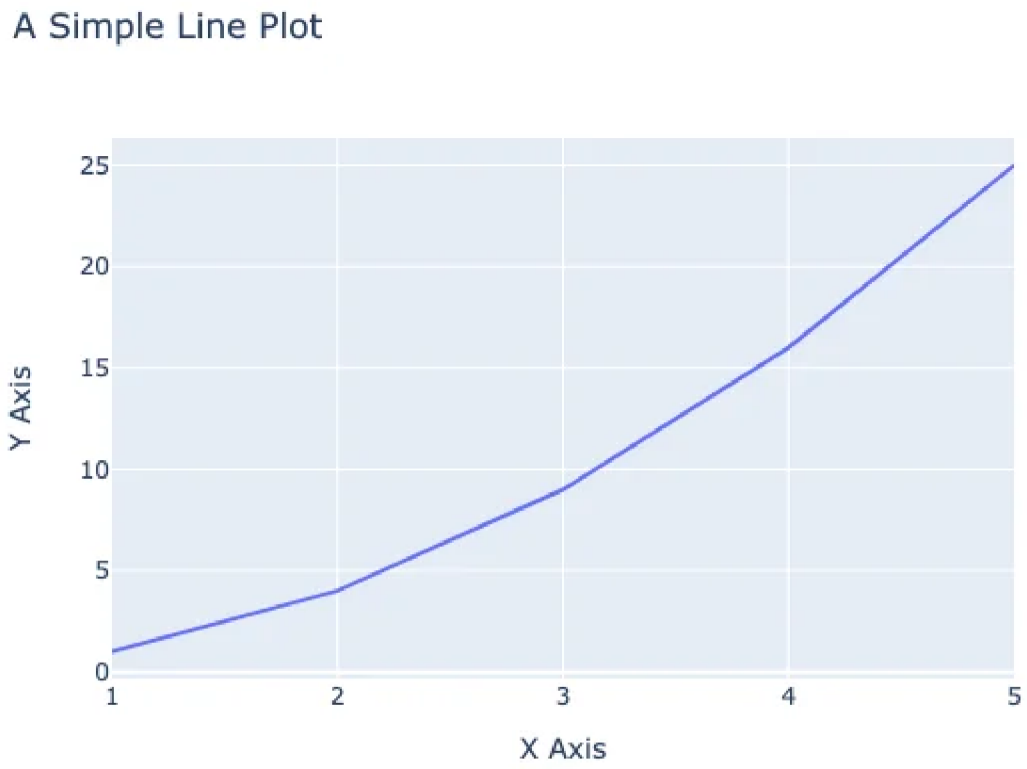
After installing the plotly as mentioned in this plotly python tutorial, using ***pip install plotly*** , let's create our first plot.

Your First plot in plotly

Let's start with a basic line chart in this python plotly tutorial:

```python
# Import necessary libraries
import plotly.express as px

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

# Create a line plot
fig = px.line(x=x, y=y, title='A Simple Line Plot', labels={'x':'X Axis', 'y':'Y Axis'})

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

When you run the above code, you'll see a line plot representing the relationship between `x` and `y`. The `title` parameter gives the plot a title, and the `labels` parameter provides custom axis labels. Here is the plot:



Creating different plots/charts in plotly

In this tutorial, we're diving deep into the world of Plotly 🌍. What you've seen is just the tip of the iceberg 🧊. As we progress, you'll be introduced to a myriad of plots – from bar charts 📊 and histograms to scatter plots and beyond. Plotly's interactive features, like zooming 🔍, panning, and hover details, make data exploration engaging and intuitive. So, gear up! 🚀 We're about to embark on an exciting journey into the heart of data visualization with Plotly. Stay with us! 🌟👩‍💻👨‍💻

Using Plotly, we have the capability to craft over 40 distinct chart types. Both `plotly.express` and `plotly.graph_objects` classes enable us to generate these plots. Let's dive in and explore some popular charts through the lens of Plotly. 📊🔍

1. Scatter Plot

A scatter plot is a type of plot that displays the relationship between two variables. It is used to visualize the correlation or relationship between two continuous variables. In a scatter plot, each data point is represented as a point on the plot with its x and y coordinates determined by the values of the two variables being plotted. Scatter plots are useful for identifying patterns or trends in the data, as well as for detecting outliers or unusual observations. They are commonly used in data analysis and statistical modeling.

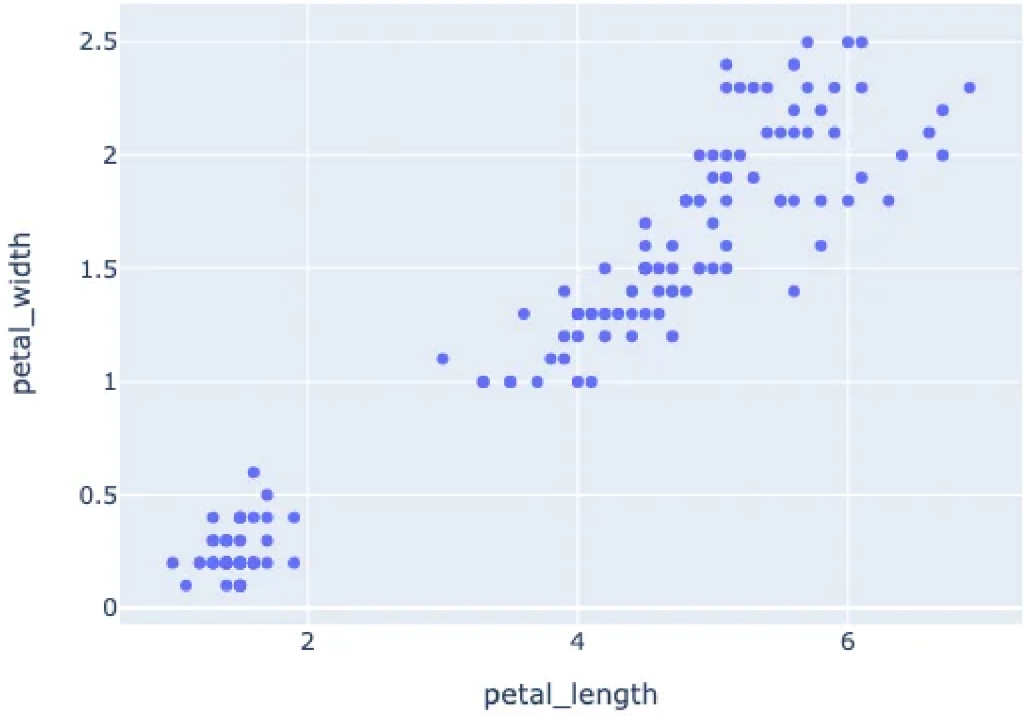
```
# import the plotly express module
import plotly.express as px

# using the iris dataset
df = px.data.iris()
```

Light Dark

```
6
7  # plotting the scatter plot
8  fig = px.scatter(df, x="petal_length", y="petal_width")
9
10 # showing the plot
11 fig.show()
```

This code imports the `plotly.express` module and loads the iris dataset from it. It then creates a scatter plot using the `px.scatter()` function with `petal_length` on the x-axis and `petal_width` on the y-axis. Finally, it shows the plot using the `fig.show()` method.

Output:

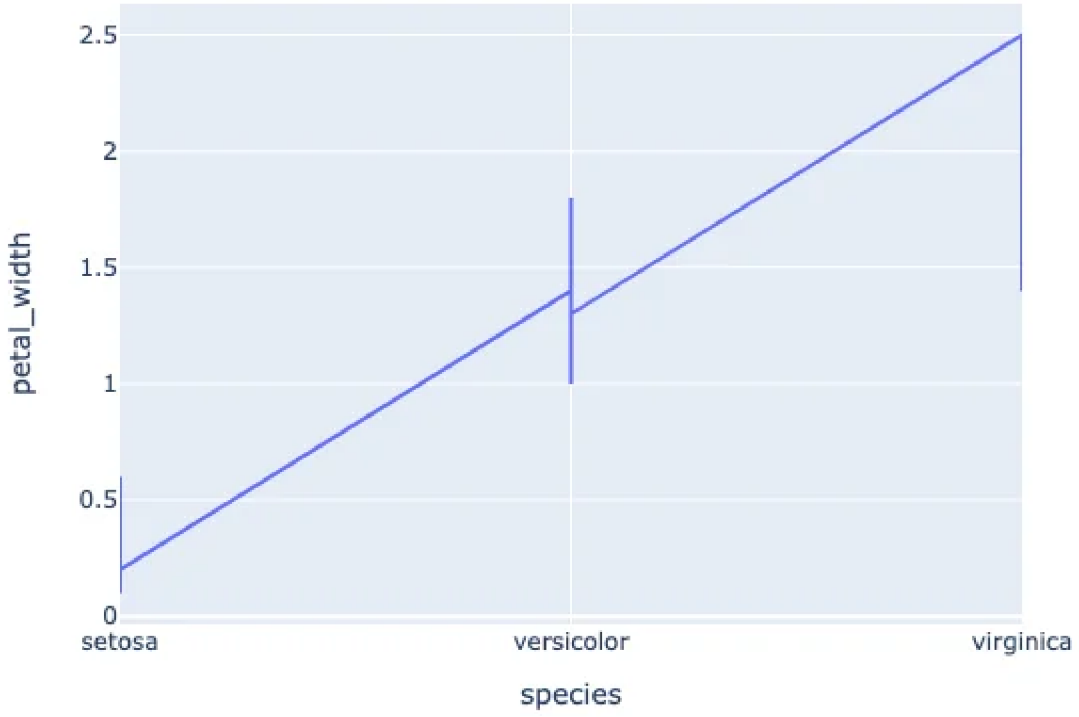


2. Line Plot or Line Chart

A line plot is a type of plot that displays the relationship between two variables as a series of data points connected by straight line segments. It is used to visualize the trend or pattern of a continuous variable over time or across different categories. In a line plot, each data point is represented as a point on the plot with its x and y coordinates determined by the values of the two variables being plotted. The points are then connected by straight lines to show the trend or pattern of the data. Line plots are useful for identifying trends, patterns, and changes in the data over time or across different categories. They are commonly used in data analysis and statistical modeling.

```python
# import the plotly express module
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the line chart
fig = px.line(df, x="species", y="petal_width")

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

Output:



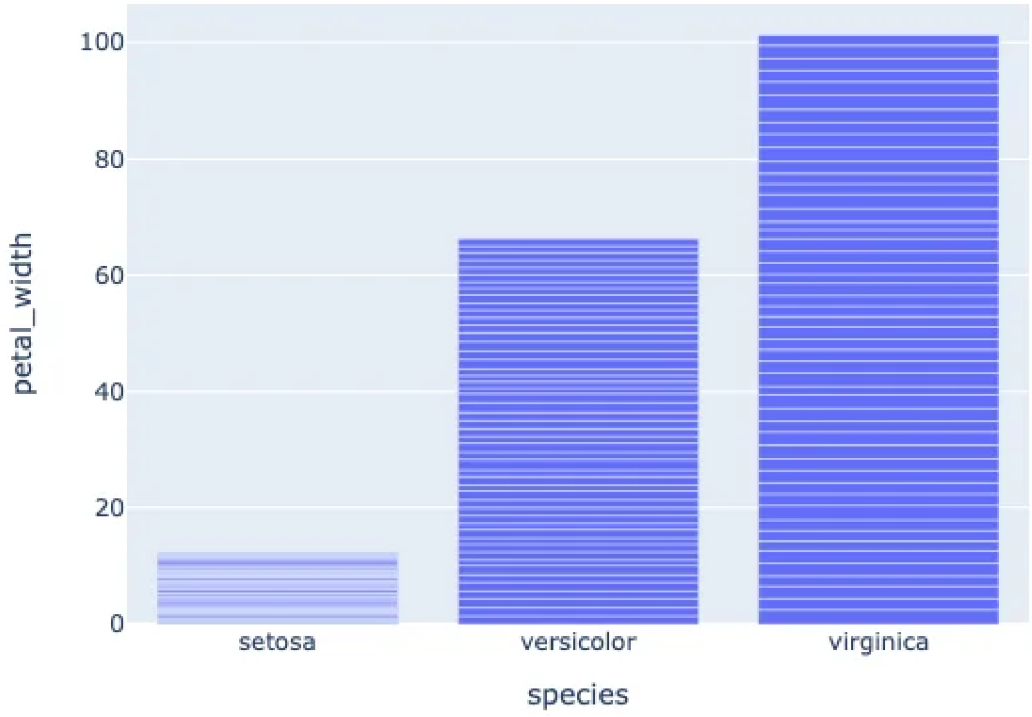
3. Bar Plot

A **bar chart** is a type of chart that displays categorical data with rectangular bars. The height or length of each bar is proportional to the value it represents. Bar charts are commonly used to compare the values of different categories or to show changes in a single category over time. In a vertical bar chart, the categories are displayed on the x-axis and the values are displayed on the y-axis. In a horizontal bar chart, the categor

the y-axis and the values are displayed on the x-axis. Bar charts are useful for visualizing data that can be divided into discrete categories, such as survey responses, sales figures, or demographic data.

```
# import the plotly express module
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the bar chart
fig = px.bar(df, x="species", y="petal_width")

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

Output:



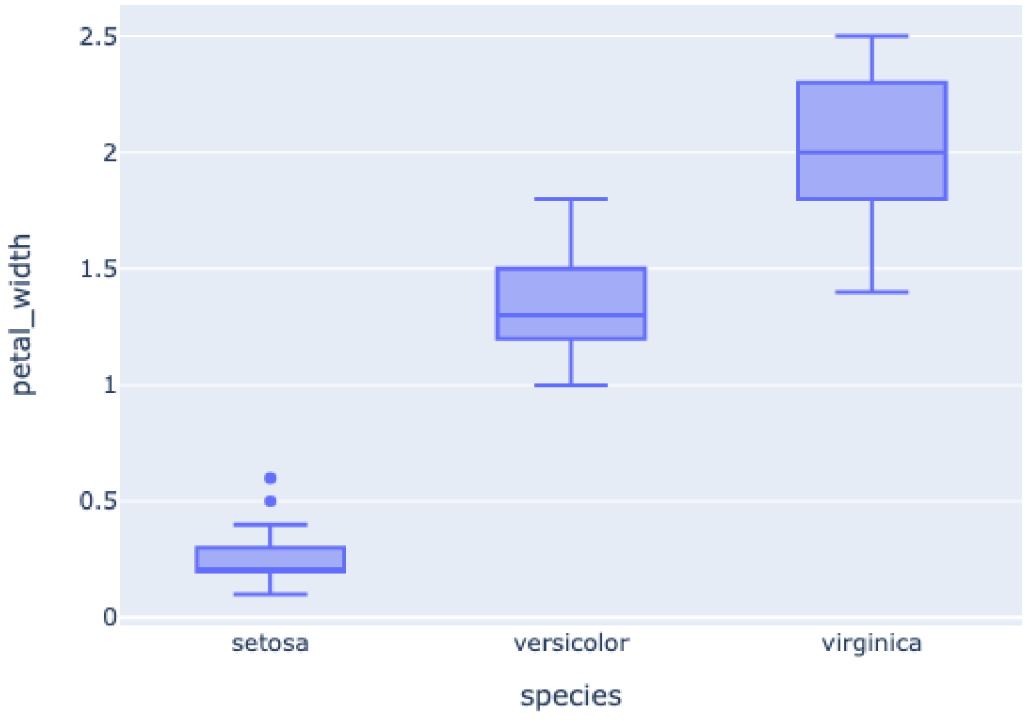
4. Box Plot

A **box plot** is a type of chart that displays the distribution of a dataset. It is used to show the median, quartiles, and outliers of the data. In a box plot, the box represents the interquartile range (IQR), which is the range between the first and third quartiles of the data. The line inside the box represents the medi

whiskers extend from the box to the minimum and maximum values within 1.5 times the IQR of the box. Any data points outside of the whiskers are considered outliers and are plotted as individual points. Box plots are useful for identifying the spread and skewness of the data, as well as for detecting outliers or unusual observations. They are commonly used in data analysis and statistical modeling.

```
# import the plotly express module
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the box plot
fig = px.box(df, x="species", y="petal_width")

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

Output:



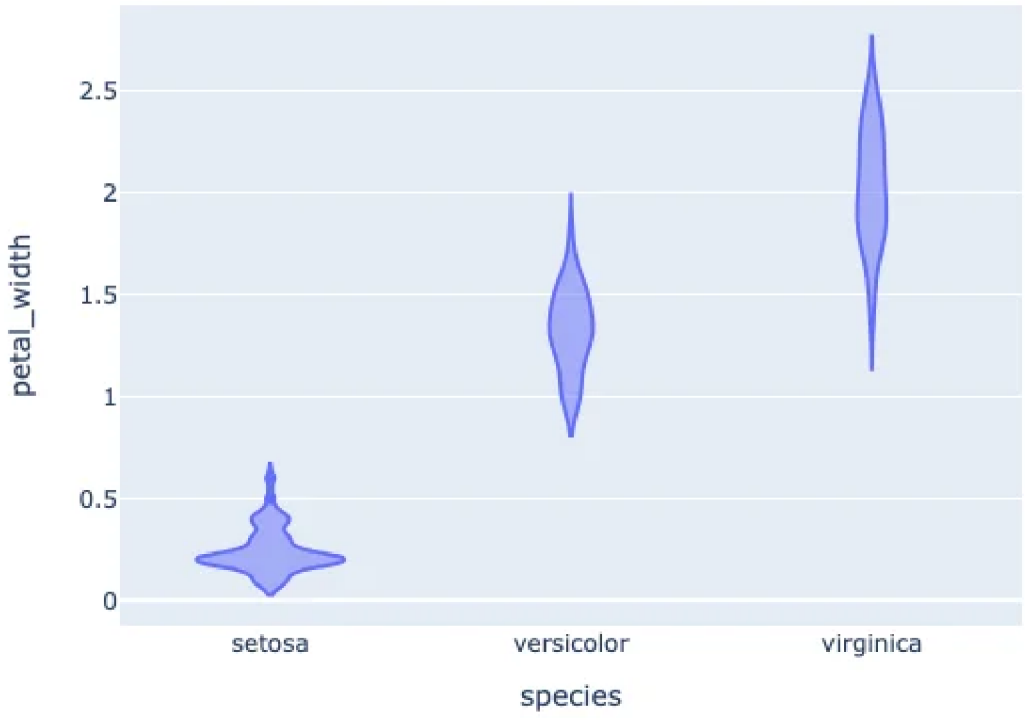
5. Violin Plot

Light Dark

Violin plots are a type of chart that display the distribution of a dataset. They are similar to box plots, but instead of showing the quartiles and outliers, they show the kernel density estimation (KDE) of the data. The KDE is a non-parametric way to estimate the probability density function of a random variable. In a violin plot, the width of the violin at a given point represents the density of the data at that point. The height of the violin represents the range of the data. Violin plots are useful for identifying the shape and spread of the data, as well as for detecting multiple modes or unusual observations. They are commonly used in data analysis and statistical modeling.

```
# import the plotly express module
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the violin plot
fig = px.violin(df, x="species", y="petal_width")

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

Output:



Light Dark

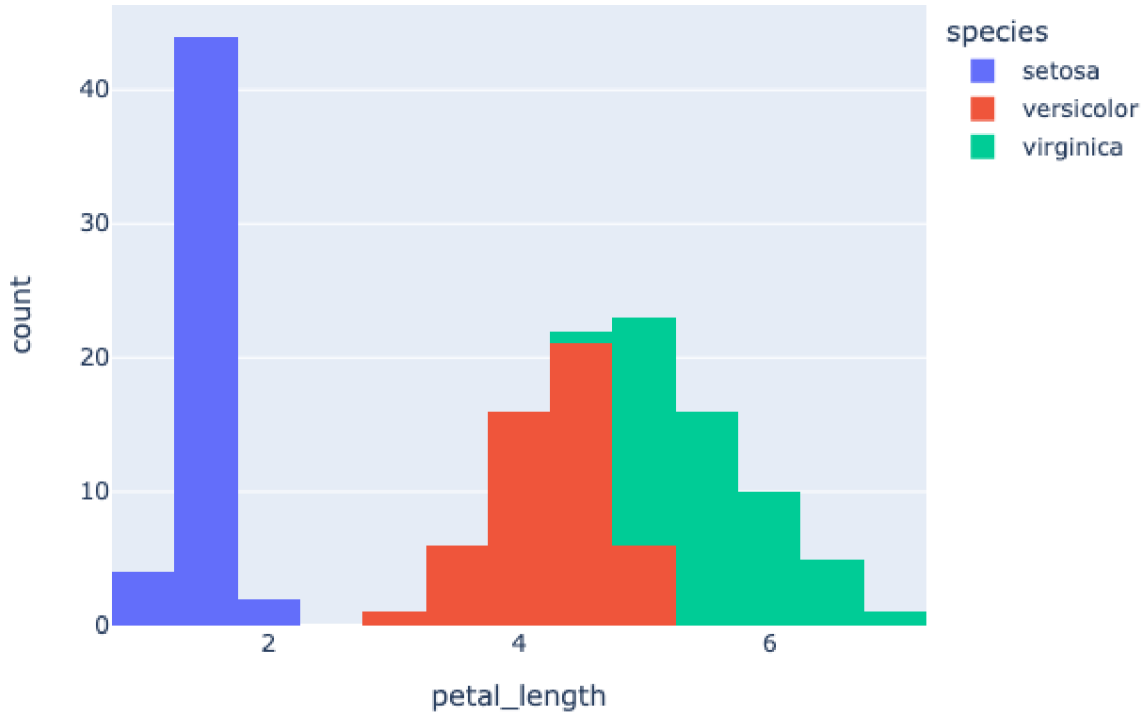
6. Histogram

A **histogram** is a type of chart that displays the distribution of a dataset. It is used to show the frequency or count of data points within a certain range or bin. In a histogram, the x-axis represents the range of values in the dataset, divided into a set of bins or intervals. The y-axis represents the frequency or count of data points within each bin. Histograms are useful for identifying the shape and spread of the data, as well as for detecting outliers or unusual observations. They are commonly used in data analysis and statistical modeling.

```
# import the plotly express module
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the histogram
fig = px.histogram(df, x="petal_length", color="species")

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

the color command in the code groups the values based on each value in ***species*** column. You can use this command in all plots.

Output:

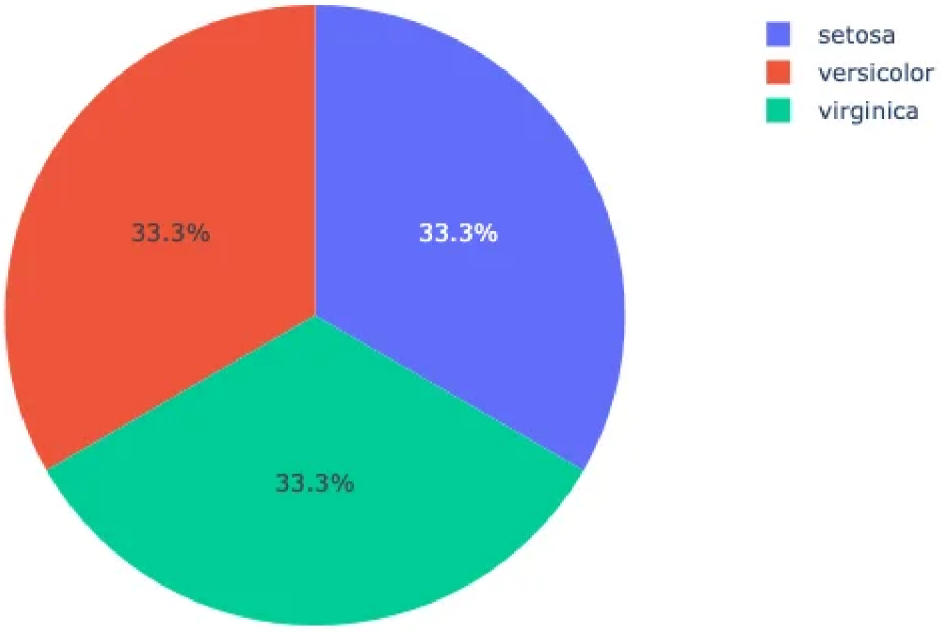


7. Pie Chart

A **pie chart** is a type of chart that displays the proportion or percentage of each category in a dataset. In a pie chart, each category is represented as a slice of the pie, with the size of the slice proportional to the value it represents. Pie charts are useful for showing the relative sizes of different categories in a dataset, but they can be difficult to read if there are too many categories or if the slices are too small. They are commonly used in data analysis and reporting.

```
# import the plotly express module
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the pie chart
fig = px.pie(df, names="species")

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

Output:



8. 3D Scatter plot

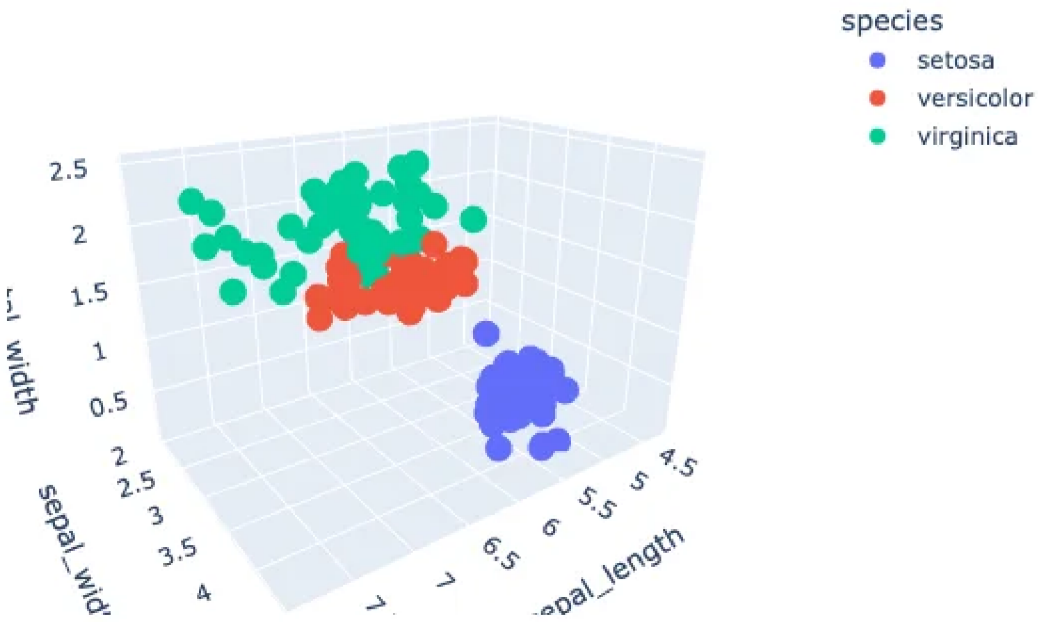
A **3D scatter plot** is useful when you want to visualize the relationship between three variables in a dataset. In a 3D scatter plot, each data point is represented as a point in 3D space, with its x, y, and z coordinates determined by the values of the three variables being plotted. This type of plot can be useful for identifying patterns or trends in the data that may not be visible in a 2D scatter plot.

```
# import the plotly express module
import plotly.express as px

# load the iris dataset from the plotly express module
df = px.data.iris()

# create a 3D scatter plot using the sepal length, sepal width, and petal width columns of
# color the points based on the species column of the dataset
fig = px.scatter_3d(df,
                    x='sepal_length',
                    y='sepal_width',
                    z='petal_width',
                    color='species')
```

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

In the code snippet you provided, a 3D scatter plot is created using the `scatter_3d()` function from Plotly Express. The `sepal_length`, `sepal_width`, and `petal_width` columns of the iris dataset are used as the x, y, and z coordinates, respectively. The `species` column is used to color the data points based on the species of the iris flower. This plot allows us to visualize the relationship between the sepal length, sepal width, and petal width of the iris flowers in 3D space, with each species represented by a different color.

Output:



9. Area chart

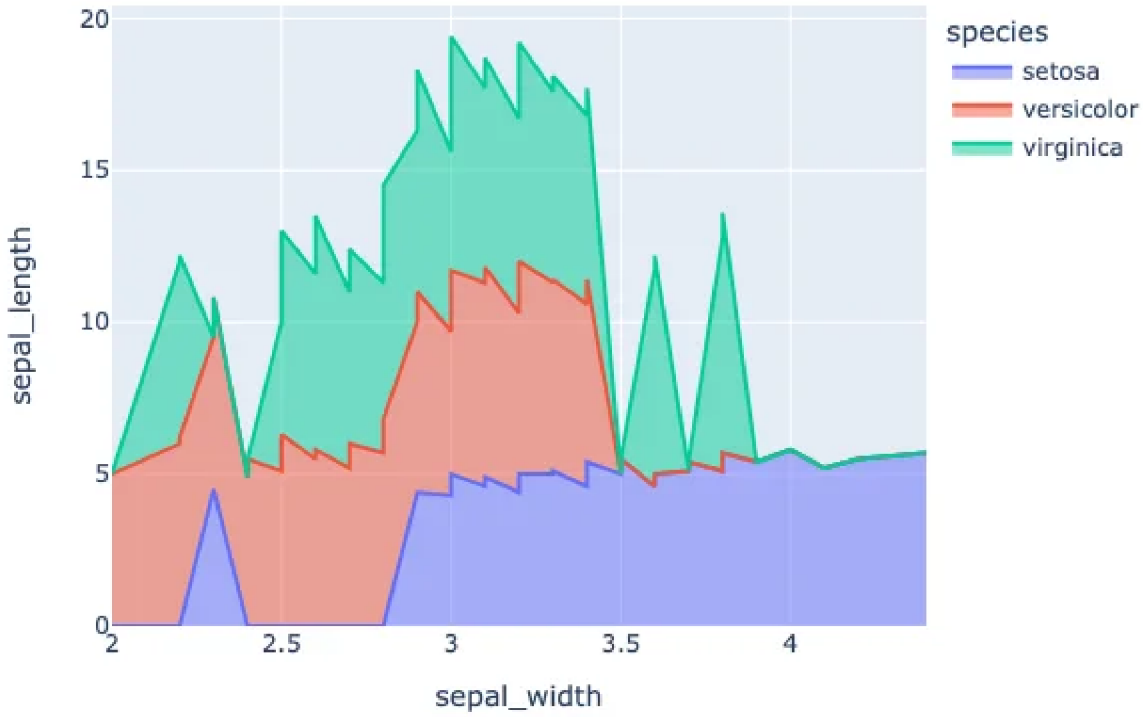
An **area chart** is a type of chart that displays the evolution of a numerical variable over time or across different categories. It is similar to a line chart, but the area between the line and the x-axis is filled with color or shading. Area charts are useful for showing the magnitude and trend of a variable, as well as for comparing the values of different categories.

```
# import the required module
import plotly.express as px

```

Light Dark

```
# load the iris dataset from the plotly express module
df = px.data.iris()

# sort the data by sepal length
df_area = df.sort_values(by=['sepal_length'])

# create an area chart using the sepal width and sepal length columns of the dataset
# color the chart based on the species column of the dataset
fig = px.area(df_area, x='sepal_width', y='sepal_length', color='species')

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

Output:

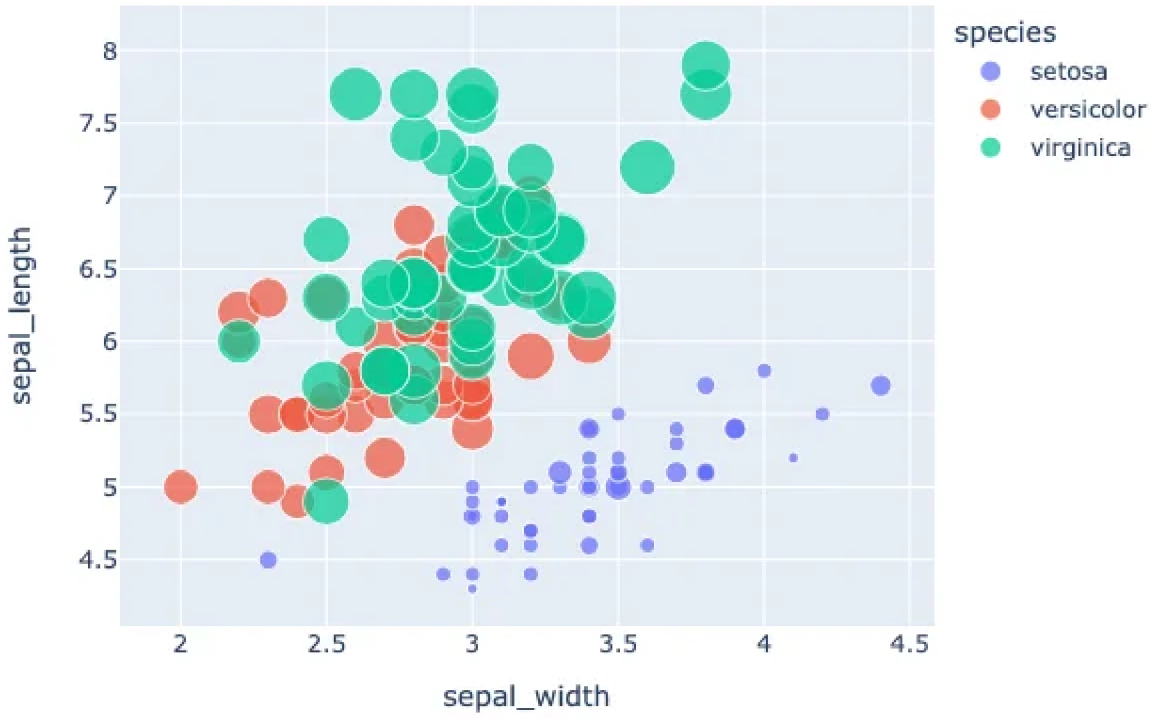


10. Bubble Chart

A **bubble chart** is a type of chart that displays three dimensions of data using the x-axis, y-axis, and size of the bubbles. It is similar to a scatter plot, but the size of the bubbles represents a third dimension of the data. Bubble charts are useful for visualizing the relationship between three variables in a dataset.

```python
# import the required module
import plotly.express as px

# load the iris dataset from the plotly express module
df = px.data.iris()

# sort the data by sepal length
df_area = df.sort_values(by=['sepal_length'])

# color the chart based on the species column of the dataset
# size the bubbles based on the petal width column of the dataset
fig = px.scatter(df_area, x='sepal_width',
                 y='sepal_length',
                 size='petal_width', color='species')

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

Output:



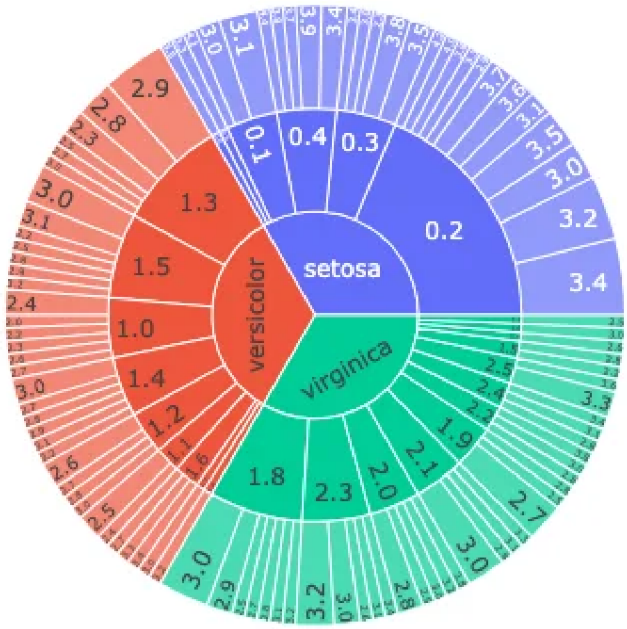
11. Sunburst Chart

Light Dark

A **sunburst chart** is useful when you want to visualize hierarchical data and the distribution of values across different categories. In a sunburst chart, each category is represented as a segment of a circle, with the size of the segment proportional to the value it represents. The segments are arranged in a hierarchical structure, with the outermost segments representing the highest level of the hierarchy and the innermost segments representing the lowest level of the hierarchy.

```
# import the required module
import plotly.express as px

# load the iris dataset from the plotly express module
df = px.data.iris()

# group the data by species, petal width, and sepal width
df_sunburst = df.groupby(['species', 'petal_width', 'sepal_width']).size().reset_index(name

# create a sunburst chart using the species, petal width, and sepal width columns of the da
fig = px.sunburst(df_sunburst, path=['species', 'petal_width', 'sepal_width'], values='coun

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

Output:



Light Dark

12. Parallel Coordinates Plot

Parallel coordinates plot (or parallel coordinates chart) is a powerful visualization technique used primarily for multi-dimensional data. Here's why we use parallel coordinates plot:

1. **High-Dimensional Data Visualization**: One of the primary uses of parallel coordinates is to visualize data that has many dimensions. Each dimension is represented by a vertical line, and data points are represented as lines that intersect these vertical lines.

2. **Discover Patterns**: It allows for the identification of patterns across multiple dimensions. Clusters of lines that follow a similar path can suggest clusters in multi-dimensional space.

3. **Spot Outliers**: Outliers can be easily identified as lines that deviate significantly from the majority of other lines.

4. **Correlation Insight**: By observing how lines move between two axes, one can infer potential correlations between dimensions. For example, if most lines slope upwards from Dimension A to Dimension B, it suggests a positive correlation.

5. **Filter and Focus**: Interactive parallel coordinates plots allow users to select ranges on specific axes to filter data, enabling a more focused analysis on subsets of data.

6. **Comparative Analysis**: It's useful for comparing multiple data points across several attributes simultaneously.

7. **Normalization Insight**: Since the scales of different dimensions can vary, the data for each dimension is often normalized. This can provide insights into how data behaves in a normalized space.

8. **Versatility**: It can be used for various data types, including continuous, discrete, and categorical data.

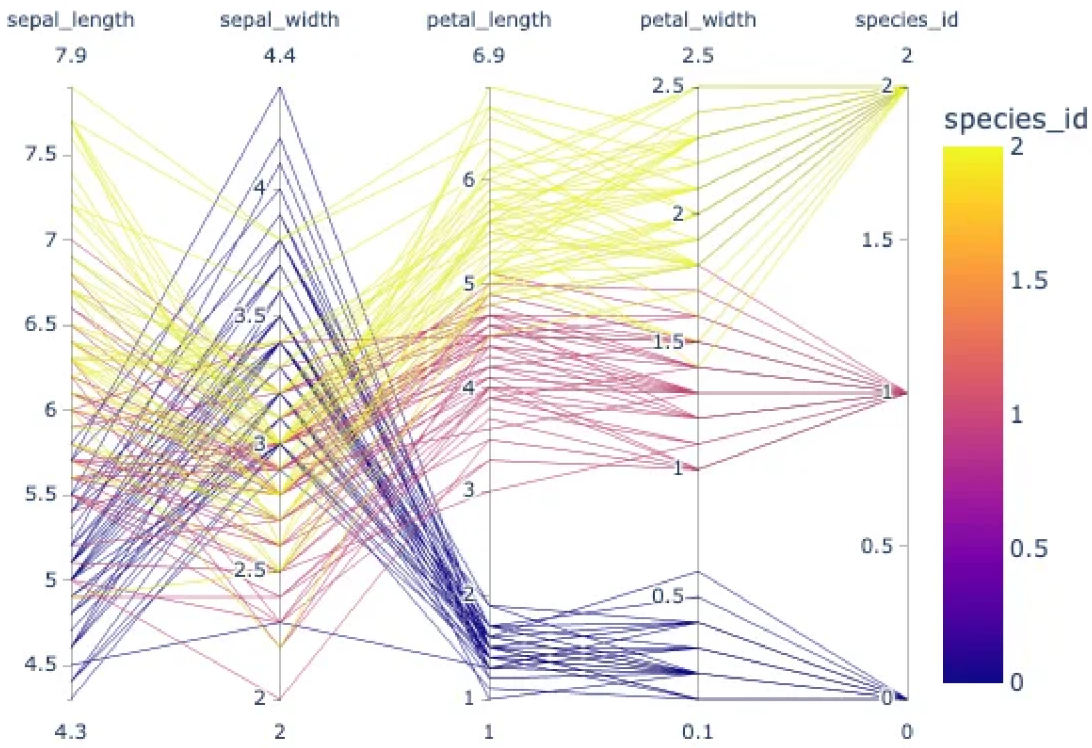
In essence, parallel coordinates plots offer a comprehensive way to visualize and analyze multi-dimensional data, making them invaluable in domains like machine learning, statistics, and data analysis where high-dimensional datasets are common.

Let's create a parallel coordinates plot using plolty in python:

```
# import the required module
import plotly.express as px

# load the iris dataset from the plotly express module

```

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

# Encode species names as numeric values (integers)
df['species_id'] = df['species'].astype('category').cat.codes

# create a parallel coordinates plot
# color the lines based on the species_id column of the dataset
# use a diverging color scale
fig = px.parallel_coordinates(df,
                              color='species_id')

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

Output:



13. Density Contour Plot

Density contour plots, often simply referred to as contour plots, are a visualization technique used to represent three-dimensional data in two dimensions. They are particularly useful for examining the relationship between two numeric variables when there are a lot of data points. Here's a breakdown of the utility and characteristics of density contour plots:

1. **Visualization of Densities**: The primary purpose of a density contour plot is to visualize areas where the density of data points is high. Essentially, it allows you to see where values are concentrated over the plot's area.

2. **Three-dimensional Data on a Two-dimensional Plane**: While the x and y axes represent two variables, the density (or frequency) of data points is represented by contour lines, effectively displaying a third dimension on a 2D plane.

3. **Contours**: These are lines (or filled areas) that connect points of equal density. The more densely packed the contour lines, the steeper the 'hill' of data points.

4. **Gradient or Color-coded**: Often, these plots use gradients or are color-coded to highlight areas of different densities. Darker shades typically indicate higher densities of data points, while lighter shades indicate sparser areas.

5. **Identification of Patterns**: Just as topographical maps allow you to identify mountains and valleys, density contour plots help identify 'peaks' or 'hotspots' where data points cluster.

6. **Noise Reduction**: By focusing on density, these plots can provide a clearer overview of patterns in large datasets by reducing the noise of individual outliers.

7. **Comparison with Histograms**: While histograms allow for the visualization of the distribution of a single variable, density contour plots provide insights into the relationship and distribution between two variables.

8. **Applications**: These plots are particularly popular in fields such as physics (e.g., potential energy surfaces in quantum mechanics), meteorology (e.g., pressure or temperature maps), geology, and any domain where the relationship between two variables and their densities is crucial.

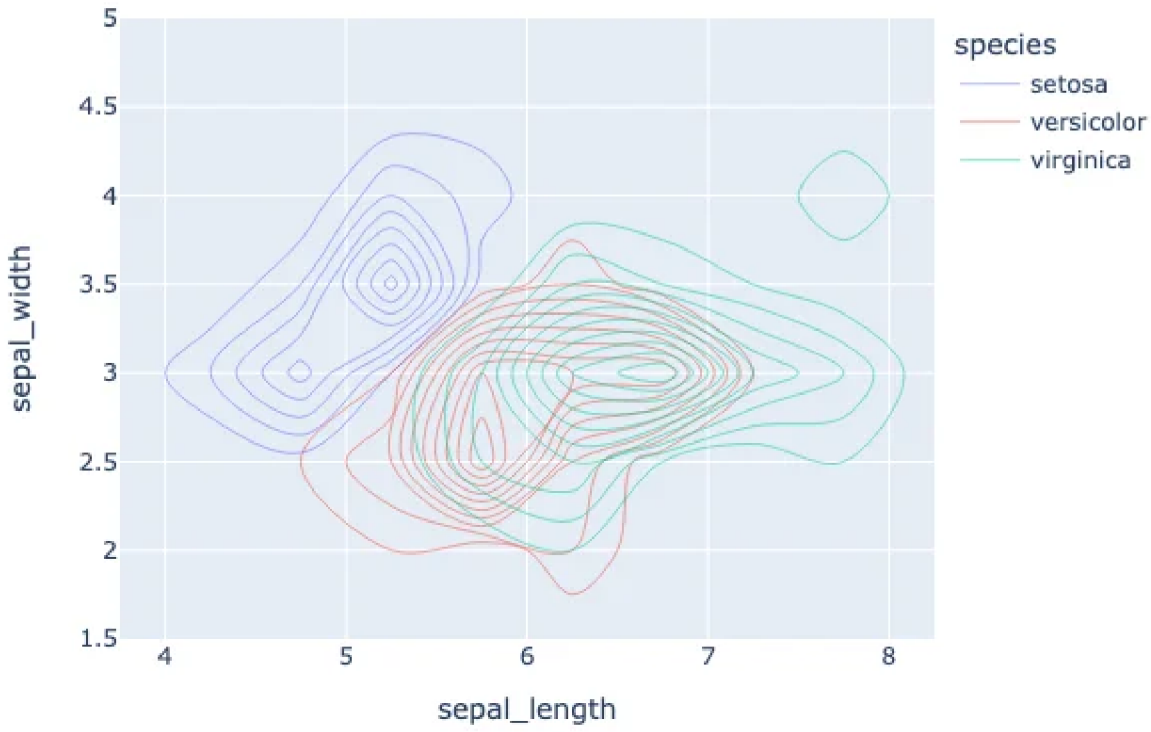
In summary, density contour plots are a valuable tool when you're interested in understanding the relationship and distribution density between two numeric variables, especially in large datasets where traditional scatter plots might be too cluttered.

Let's create a Density Contour plot using plotly python:

```
# Importing libraries
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the density contour plot
fig = px.density_contour(df, x='sepal_length',
```

```
                          y='sepal_width',
                          color='species')
# Show the plot
fig.show()
```

Output:



14. Ternary Plot

A ternary plot, also known as a ternary diagram or triangle plot, is a triangular diagram used to visualize the proportions of three variables that must sum to a constant, typically 100% (like in the case of components of a mixture). It's a unique and specialized plot that provides insights into the interrelationships between the three variables. Here's more about the ternary plot:

1. **Triangular Coordinate System**: The plot has three axes, each representing one of the three variables. These axes meet at a central point, forming an equilateral triangle. Each corner of the triangle represents the pure component of one of the three variables.

2. **Normalized Scale**: The scale for each axis typically runs from 0 to 1, with data points plotted inside the triangle based on the proportions of each of the three variables they represent.

3. **Data Points**: A data point inside the triangle represents the ratios of the three variables. A point near one of the triangle's corners indicates a high proportion of the respective variable.

4. **Barycentric Coordinates**: Positions within the triangle are determined using barycentric coordinates, which consider each triangle corner as a reference.

5. **Applications**:
    - **Geology & Petrology**: Used to represent the proportions of three minerals in a rock sample.
    - **Chemistry**: To depict the composition of mixtures, such as in phase diagrams.
    - **Ecology**: For species composition in ecosystems.
    - **Genetics**: To show allele frequencies in populations.

6. **Visual Enhancements**: Color-coding or gradient scales can be used to represent an additional variable or to illustrate the density of data points within regions of the ternary plot.

7. **Comparison with Other Plots**: Unlike scatter plots or contour plots which visualize relationships between two variables, ternary plots allow for the simultaneous visualization of three-part compositional data.

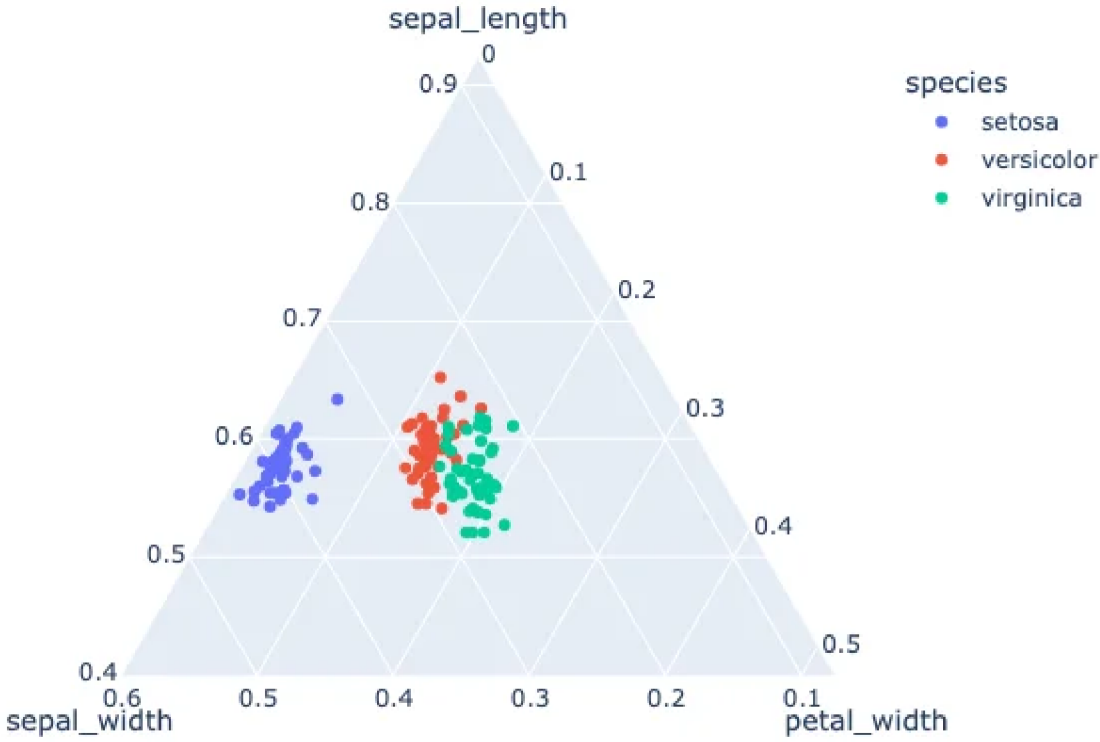
8. **Interpretation**: Reading a ternary plot can be less intuitive compared to some other plot types, and it might require some experience. However, it offers a unique way to capture and analyze the complexity of systems defined by three interrelated variables.

In essence, ternary plots are a powerful tool for visualizing and analyzing the relationships and compositions of three-part systems. They condense complex tri-variate information into a two-dimensional, triangular format, making it possible to see the interactions and relative proportions of three variables at a glance.

Let's create a ternary plot using plotly python on iris dataset:

```
# Importing libraries
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the Scatter ternary plot
fig = px.scatter_ternary(df, a='sepal_length',
                         b='sepal_width',
                         c='petal_width',
                         color='species')
# Show the plot
fig.show()
```

Output:



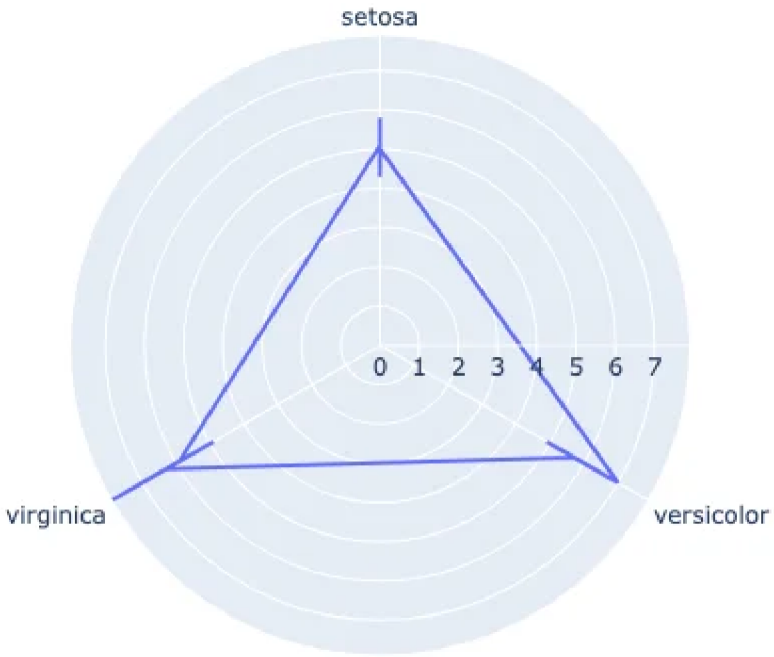
15. Polar/Radar/Spider Chart

A **polar chart**, often referred to as **a radar or spider chart**, is a graphical method of displaying multivariate data in the form of a two-dimensional chart. Each variable is represented on a separate axis that starts from a common center point and extends outwards.

```
# Importing libraries
import plotly.express as px

# using the iris dataset
df = px.data.iris()
# create Polar or Radar chart
fig = px.line_polar(df, r='sepal_length',
                    theta='species',
                    line_close=True)
# Show the plot
fig.show()
```

Output:

Light Dark



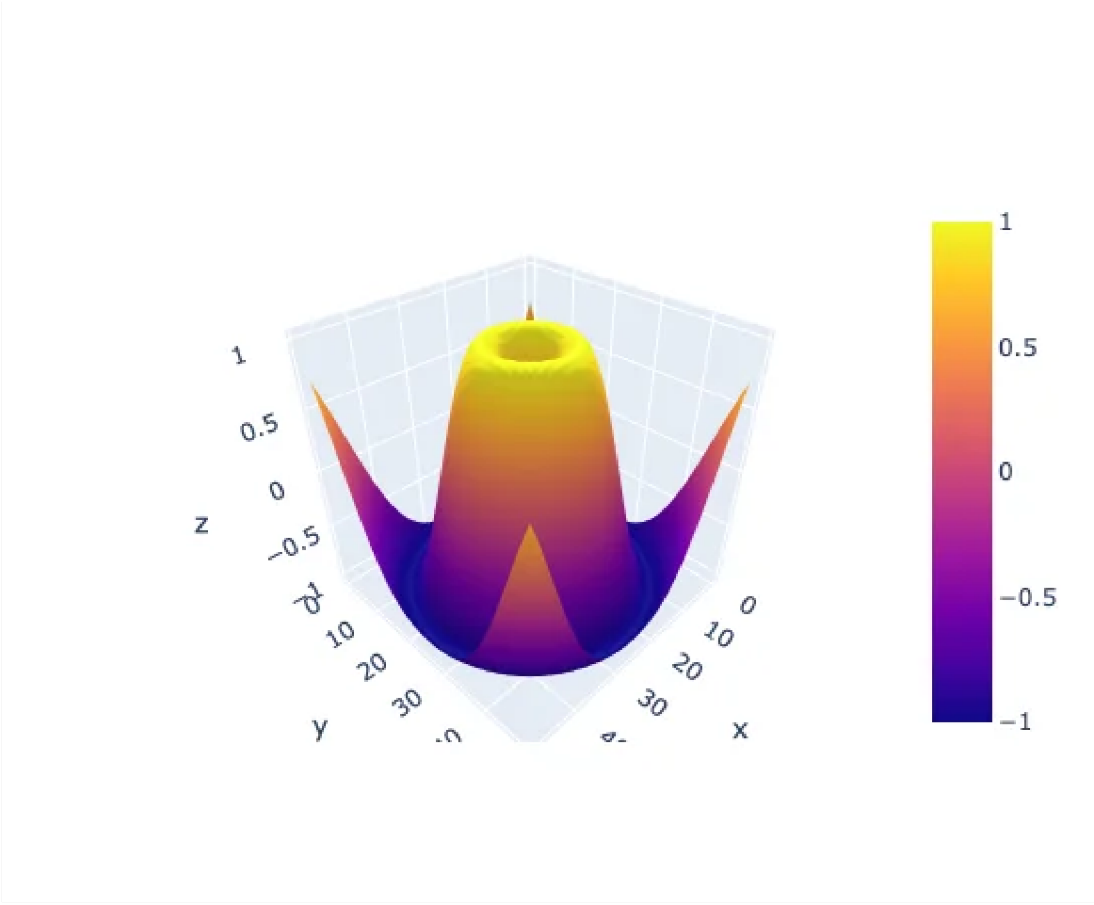
16. 3D surface plot

A **3D surface plot** visualizes three-dimensional data by representing relationships between three continuous variables. Each axis of the plot corresponds to one of the variables, while the surface height and color variations depict value changes. It's especially useful for understanding complex datasets and exploring topographical patterns, such as peaks and valleys, in the data landscape.

```
# import the required module
import plotly.graph_objects as go
import numpy as np

# Create a dummy data
x = np.linspace(-5,5,50) # create an array of 50 evenly spaced values between -5 and 5
y = np.linspace(-5,5,50) # create an array of 50 evenly spaced values between -5 and 5
X,Y = np.meshgrid(x,y) # create a grid of x and y values
Z = np.sin(np.sqrt(X**2 + Y**2)) # calculate the value of z for each point on the grid

# Create a surface plot
fig = go.Figure(data=[go.Surface(z=Z)]) # create a surface plot using the calculated z valu
fig.show() # display the plot
```

Light Dark

Output:



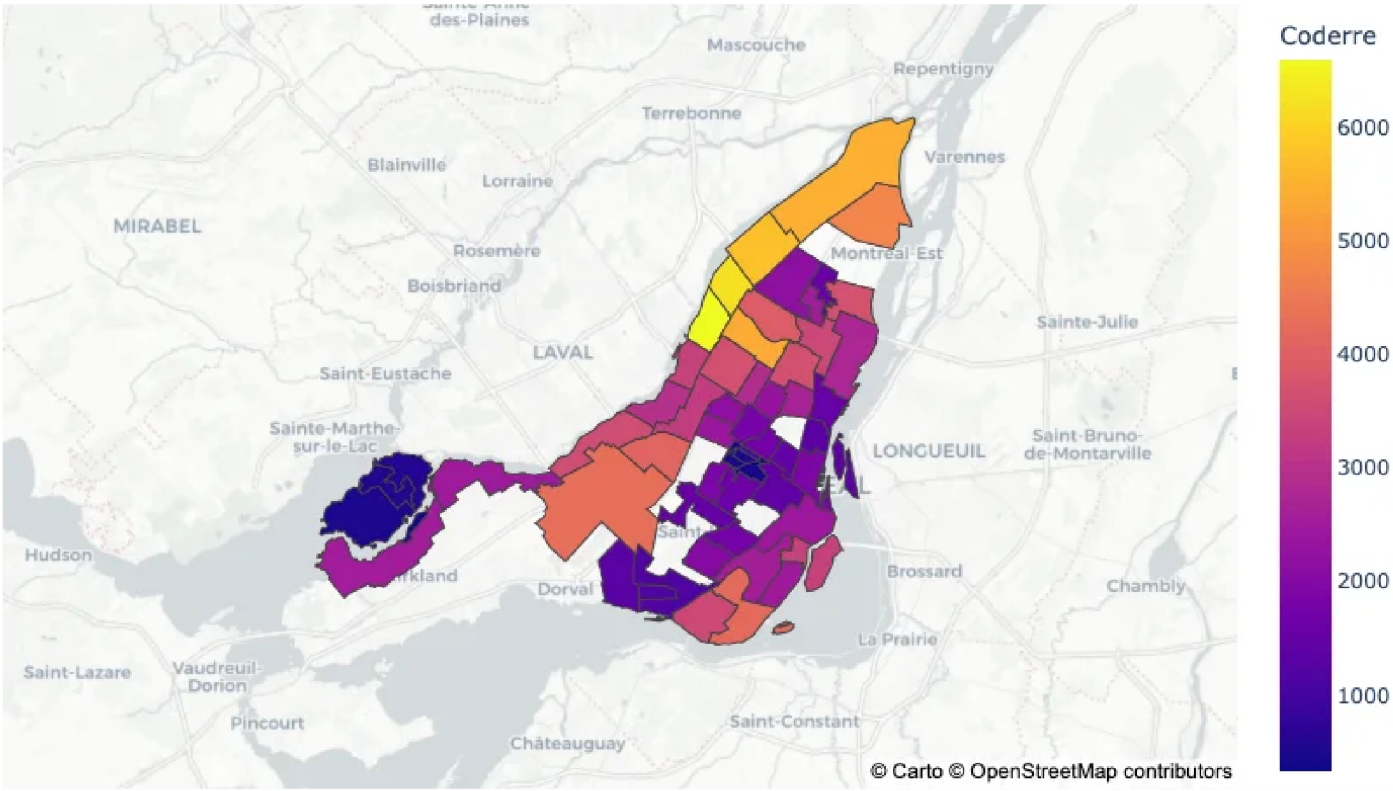
17. Geographical map

Geo maps, also known as **choropleth maps**, are a type of map that displays data using different colors or shades to represent different values or levels of a variable across different geographic regions. In a geo map, each region is represented as a polygon or shape, with the color or shade of the polygon determined by the value of the variable being plotted. Geo maps are useful for visualizing the distribution of data across different geographic regions, such as countries, states, or election districts. They are commonly used in data analysis, reporting, and visualization.

```
# import the necessary libraries
import plotly.express as px

# load the sample election data
df = px.data.election()

# load the geojson file for the election districts
geojson = px.data.election_geojson()

# create a choropleth map using plotly
fig = px.choropleth_mapbox(df, geojson=geojson, color="Coderre",
                           locations="district", featureidkey="propertie.
```

Light Dark

```
                          center={"lat": 45.5517, "lon": -73.7073},
                          mapbox_style="carto-positron", zoom=9)

# update the layout of the map
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})

# display the map
fig.show()
```



18. Animated GEO Map in plotly

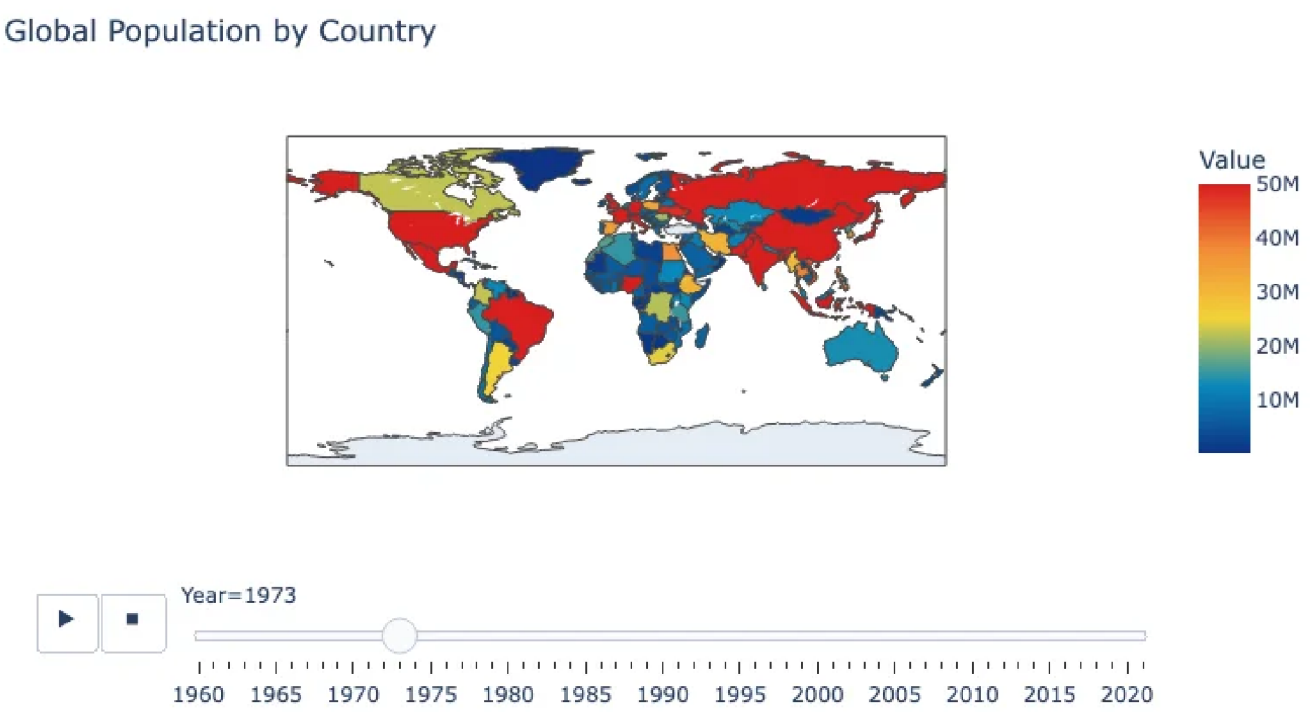
We can also create the animated plots with plotly using express module.

The purpose of this code is to create a choropleth map using Plotly Express to visualize the global population by country over time.

```
# import the necessary libraries
import pandas as pd
import plotly.express as px

# load the population data from a csv file hosted on GitHub
df = pd.read_csv('https://raw.githubusercontent.com/datasets/population/main/data/populatio

# create a choropleth map using plotly
fig = px.choropleth(df, locations='Country Name', locationmode='country
                    animation_frame='Year', range_color=[100000, 50000000
```

Light Dark

```
                    color_continuous_scale='portland',
                    title='Global Population by Country')

# display the map
fig.show()
```

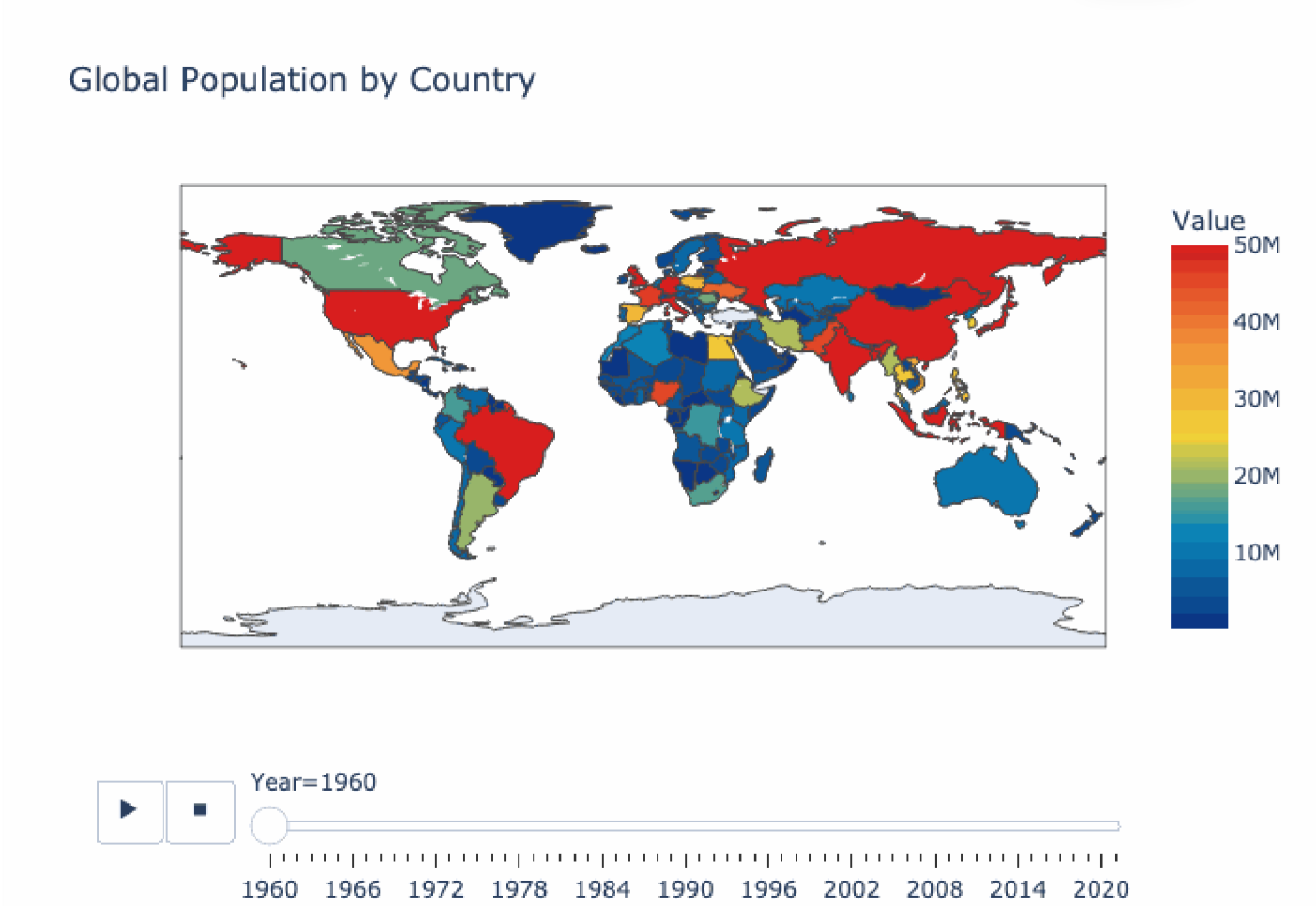


Saving plots as .gif file

Saving the plotly animated plot as ***gif*** file can be done as follows:

```
# import the necessary libraries
import plotly.express as px
import pandas as pd
import numpy as np
import io
import PIL

# load the population data from a csv file hosted on GitHub
df = pd.read_csv('https://raw.githubusercontent.com/datasets/population/main/data/populatio

# create a choropleth map using plotly
```

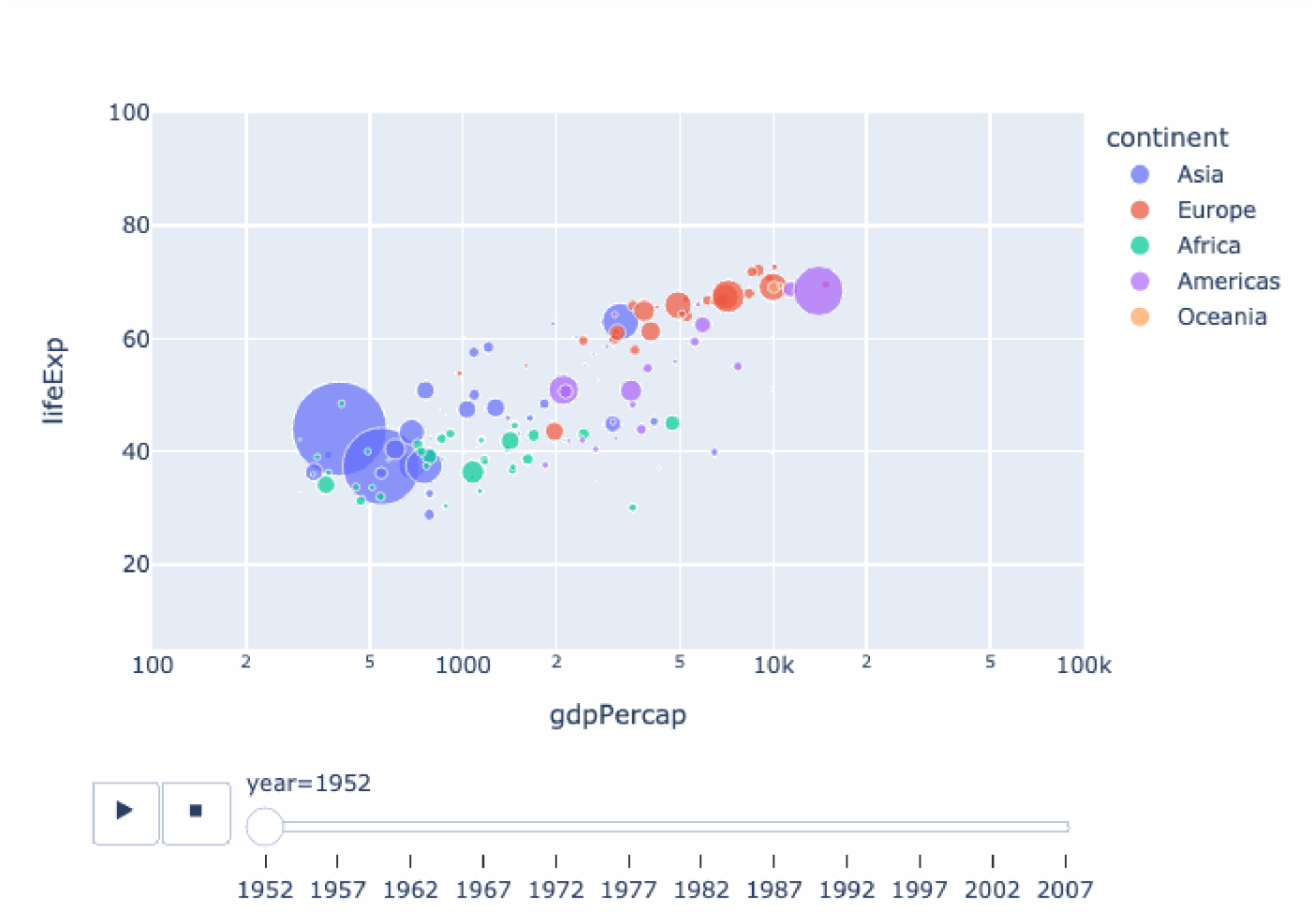
```
fig = px.choropleth(df, locations='Country Name', locationmode='country names', color = 'Va
                    animation_frame='Year', range_color=[100000, 50000000],
                    color_continuous_scale='portland',
                    title='Global Population by Country')

# save the animated map as a gif file

# generate images for each step in animation
frames = []
for s, fr in enumerate(fig.frames):
    # set main traces to appropriate traces within plotly frame
    fig.update(data=fr.data)
    # move slider to correct place
    fig.layout.sliders[0].update(active=s)
    # generate image of current state
    frames.append(PIL.Image.open(io.BytesIO(fig.to_image(format="png", scale=2))))

# create animated GIF
frames[0].save(
        "world_population_in_recent_years.gif",
        save_all=True,
        append_images=frames[1:],
        optimize=True,
        duration=500, # milliseconds per frame
        loop=0, # infinite loop
        dither=None  # Turn off dithering
    )
```



19. Animated bubble chart

```python
# import the necessary libraries
import plotly.express as px
import pandas as pd
import numpy as np
import io
import PIL

# load the gapminder data from plotly
df = px.data.gapminder()

# create a choropleth map using plotly
fig = px.scatter(df, x= "gdpPercap",
            y = "lifeExp",
            size= "pop", color= "continent",
            animation_frame='year', animation_group="country",
            log_x=True, size_max=55, range_x=[100,100000], range_y=[5,100])


# save the animated plot as a gif file

```

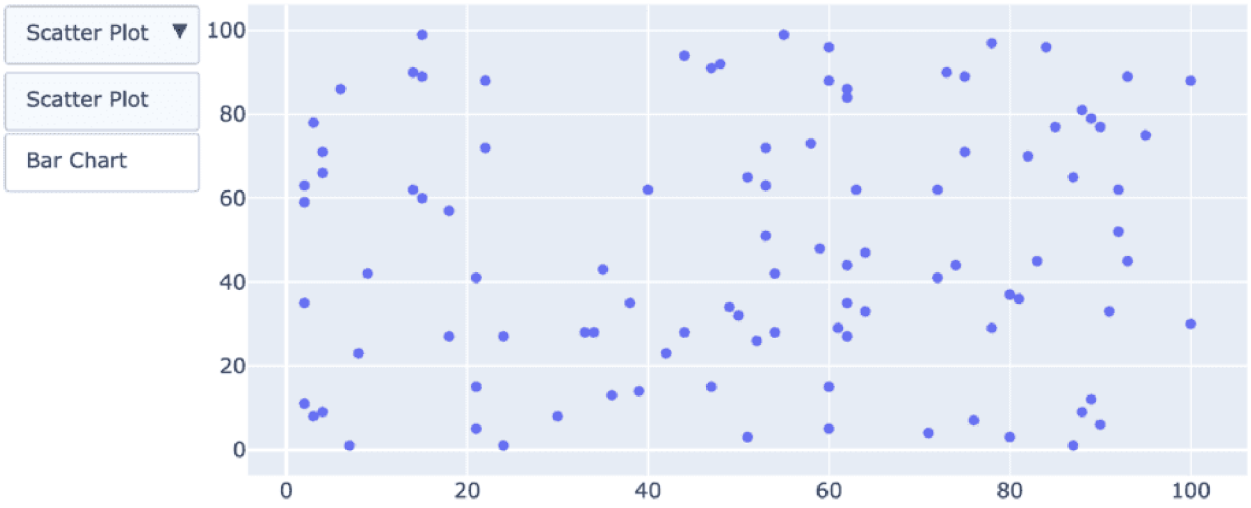
```
# generate images for each step in animation
frames = []
for s, fr in enumerate(fig.frames):
    # set main traces to appropriate traces within plotly frame
    fig.update(data=fr.data)
    # move slider to correct place
    fig.layout.sliders[0].update(active=s)
    # generate image of current state
    frames.append(PIL.Image.open(io.BytesIO(fig.to_image(format="png", scale=1))))

# create animated GIF
frames[0].save(
        "Animated Bubble Plot using plotly.gif",
        save_all=True,
        append_images=frames[1:],
        optimize=True,
        duration=500, # milliseconds per frame
        loop=0, # infinite loop
        dither=None  # Turn off dithering
    )
```



20. Add drop down button in plotly

Light Dark

```
import plotly.graph_objects as px
import numpy as np


# creating random data through randomint
# function of numpy.random
np.random.seed(42)

# Data to be Plotted
random_x = np.random.randint(1, 101, 100)
random_y = np.random.randint(1, 101, 100)

plot = px.Figure(data=[px.Scatter(
    x=random_x,
    y=random_y,
    mode='markers',)
])

# Add dropdown
plot.update_layout(
    updatemenus=[
        dict(
            buttons=list([
                dict(
                    args=["type", "scatter"],
                    label="Scatter Plot",
                    method="restyle"
                ),
                dict(
                    args=["type", "bar"],
                    label="Bar Chart",
                    method="restyle"
                )
            ]),
            direction="down",
        ),
    ]
)

plot.show()
```

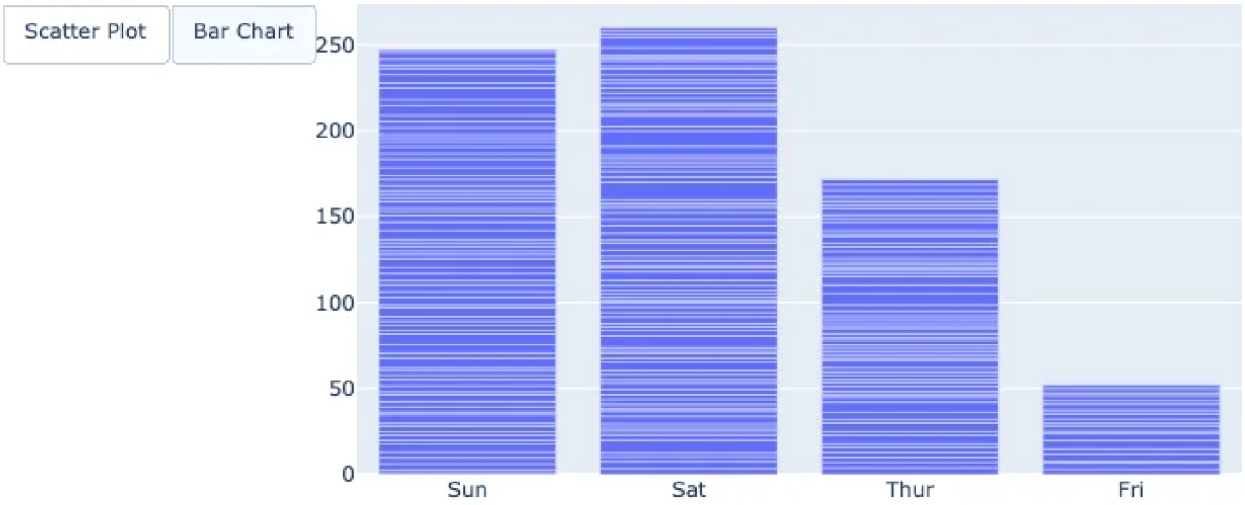


21. Add buttons

```
import plotly.graph_objects as px
import pandas as pd
import seaborn as sns

# reading the database
data = sns.load_dataset('tips')


plot = px.Figure(data=[px.Scatter(
    x=data['day'],
    y=data['tip'],
    mode='markers',)
])

# Add dropdown
plot.update_layout(
    updatemenus=[
        dict(
            type="buttons",
            direction="left",
            buttons=list([
                dict(
                    args=["type", "scatter"],
                    label="Scatter Plot",
                    method="restyle"
                ),
```

Light Dark

```
                dict(
                    args=["type", "bar"],
                    label="Bar Chart",
                    method="restyle"
                )
            ]),
        ),
    ]
)

plot.show()
```



Conclusion

Plotly stands out as a pivotal library in the Python data visualization ecosystem, primarily due to its emphasis on interactive and animated plots. In the age of digital data exploration, static charts often fall short in conveying the depth and nuances of complex datasets. Plotly bridges this gap by allowing users to engage with visualizations, drilling down into details, panning across dimensions, and even animating transitions to understand temporal changes. Its wide array of customizable chart types caters to both conventional and specialized visualization needs. There are serval other tutorials on python plotly use such as one by geeksforgeeks also many other plotly articles given at the following link.

Data Science data visualization python

Furthermore, its seamless integration with web applications, courtesy of Dash, positions Plotly as not just a visualization tool, but as a comprehensive platform for data-driven web applications. In essence, Plotly's capability to create intuitive, interactive, and animated visualizations elevates data storytelling, making it an indispensable tool for data professionals aiming to communicate insights effectively.

In this python plotly tutorial, we have seen 20 different types of plots, their use and the codes to plot them. I hope you like the way we explained, please write down in the comment section, what you want to learn next.

Other resources on our website

You can also attempt quizzes here.

- Data Visualization: Unlocking the insights of data

PREVIOUS
Data Visualization Quiz

NEXT
Mastering the Art of Data Preprocessing

Search

Search

Recent Blog Posts

- A Guide to Prompt Engineering for Data Scientists & AI Developers in Pakistan | Codanics
- An Analytical Perspective on Gold Rates: USD to PKR Market Trends in Pakistan
- The Importance of MLOps for Data Science Students
- What is MLOps? A Beginner's Guide | Codanics
- Top 10 benefits of using vscode as your code editor or IDE

Muhammad Fahad Bashir

May 13, 2025 at 9:59 am

Your comment is awaiting moderation. This is a preview, your comment will be visible after it has been approved.

it's great cheat sheat to undertsand plotly

Reply

Danish Azeem

May 16, 2024 at 3:38 pm

very nice blog on Plotly; it covers all the basics. Very helpful

Reply

FEROZ SHAH

January 10, 2024 at 2:52 pm

it was a little lengthy but suuper informative made every thing very much easy by

Reply

Muneer Ahmed

January 4, 2024 at 11:25 am

I wanted to thank you for your hard work and dedication

Reply

Aftab Ahmad

December 9, 2023 at 9:51 am

Baba g lambi umar pao tosi

Reply

Dr. Junaid Iqbal Bhatti

November 12, 2023 at 11:46 pm

Great Job Dr. shb

Reply

Quratul Ain

November 8, 2023 at 4:27 pm

Alhumdulillah nice explain

Reply

Javed Ali

November 6, 2023 at 4:33 am

سلام علیکم ورحمة الله وبرکاته
آپ کا پڑھانے کا طریقہ بہت ہی اعلیٰ ہے
چیزوں کو اتنی مرتبہ دھرایا جاتا ہے کہ وہ اچھی طرح زہن نشین ہوجاتیں ہیں
یہ بلاگ بہت ہی کمال کا ہے
تمام قسم کے پلاٹ بہت ہی اچھی طرح سمجھ میں آگئے ہیں
الله کریم آپ کو دونوں جہان کی بھلائیاں عطا کرے آمین

Reply

Furqan Malik

November 5, 2023 at 2:38 am

good blog

Reply

zeeshan younas

November 4, 2023 at 12:37 am

Sir ge bht axha blog tha bht si confusion clear ho gi is sy

Reply

Moavia Hassan

November 2, 2023 at 9:39 pm

very nice blog on plotly it covers all the basics and fundamentals of making plots using plotly

Reply

Mudassar Hussain

November 2, 2023 at 7:53 pm

Informative and supportive thanks. Sir doctor aammar

Reply

Mudassar Hussain

November 2, 2023 at 7:50 pm

I completed my 70persnt problems in this blog

Reply

Muhammad Shariq

November 2, 2023 at 7:06 pm

It’s describe in amazing way….

Reply

Kanwalzeb

November 2, 2023 at 4:06 pm

Assalamualaikum sir very informative blogs. It will help me in this course. jazakAllah

Reply

Mohsin Shareef

November 2, 2023 at 1:14 pm

Present Sir, Mohsin Shareef
Discord # mohsinshareef_22482_34614

Reply

Muhammad Naeem

November 2, 2023 at 1:12 pm

very informative and amazing method of teaching

Reply

Rizwana Shafi

November 2, 2023 at 10:05 am

it's very helpful blog

Reply

Muhammad Mursaleen

November 2, 2023 at 1:43 am

For me, this is the best way of teaching. I didn't see it before. This is the first time to taken any course take it is a very energetic way to learn. May Allah give you health to fulfill 6 months. Ameen
for me. I also parallel with python ka Chilla 2023. it's going to be great and waiting for your next Blog.
For Data Preprocessing.

Reply

Maria Nadeem

November 2, 2023 at 12:35 am

This informative blog provides step-by-step guidance on creating effective and interactive plots with Plotly. It clarifies Plotly plot concepts, making it a valuable resource for understanding and mastering plot creation.

Reply

Nimra Ishaq

November 2, 2023 at 12:28 am

Amazing and helpful blog.

Reply

Shafat Hussain Khan

November 1, 2023 at 11:12 pm

mashallah
Very helpful
3D surface plots and geographical maps.amazing

Reply

Danish Shafiq Khan

November 1, 2023 at 11:00 pm

thank u sir and present sir 🙄

Reply

Zubair Ahmad

November 1, 2023 at 10:36 pm

Present Sir,
by Zubair Ahmad
Whatsap # 03012664962
Discord # zubair_ahmad

Reply

komal Baloch

November 1, 2023 at 9:54 pm

very helpful sir

Reply

Muhammad Haroon

November 1, 2023 at 9:52 pm

It is amazing to learn plotly, for begginers

Reply

Saman Fatima

November 1, 2023 at 9:51 pm

Detailed Article thanks alot for teaching us sir 🙄

Reply

Gulafsha Khan

November 1, 2023 at 9:49 pm

Informative and amazing, enjoying practicing

Reply

Farwa Khalid

November 1, 2023 at 6:12 pm

Read this blog and put it into practice. This comprehensive blog has greatly enriched my learning experience.

Reply

Farwa Khalid

November 1, 2023 at 6:11 pm

“Read this blog and put it into practice. This comprehensive blog has greatly enriched my learning experience.”

Reply

Reyan

November 1, 2023 at 11:15 am

I appreciate your teaching style. You are very organized, patient, and supportive. You provide helpful feedback and guidance. You make learning fun and interactive. You are the best teacher I ever had.

Reply

M Usman

November 1, 2023 at 10:23 am

Mind-blowing article to understand plotly library

Reply

Mehak Iftikhar

October 31, 2023 at 11:11 pm

Amazing and very informative

Reply

Mahboob

October 31, 2023 at 9:02 pm

very nice, helpful and understandabe. May God Bless you

Reply

Muhammad Abdullah

October 30, 2023 at 8:43 pm

Ma sha Allah ❤️ Every Plot Is Well Explained ..

Reply

Amber Fatima

October 30, 2023 at 8:36 pm

The examples provided in this blog are very helpful in getting started with the library. I found the plotly library to be very useful for creating interactive and visually appealing plots and charts in Python. The library offers a wide range of chart types, from simple scatter plots to complex 3D surface plots and geographical maps.

Reply

Sidra Akbar

October 30, 2023 at 8:02 pm

The blog provided an informative overview of various plotting strategies across different libraries. It helped create an understanding of the common plotting concepts and techniques, while also highlighting the similarities and differences in how these concepts are implemented in diverse visualization packages. The comprehensive discussion of plotting helps establish a solid foundation for utilizing various data visualization tools.

Reply

Aftab Ahmad Khan

October 30, 2023 at 6:30 pm

Bundle of information. This blog created an understanding of various plots. Of course, the plotting strategies are similar in different libraries.

Reply

Mohammad Sohail

October 30, 2023 at 3:51 pm

Mind blowing way of teaching, to the point & simple coding structure

Reply

Imran Ur rehman

October 30, 2023 at 10:02 am

mashallah. to the point guidance as usual.
Kindly inform that on which dashboard we will be working in future ? Tableau or power B

Reply

Rashad Masud

October 30, 2023 at 8:30 am

Very helpful

Reply

Shahid Umar

October 30, 2023 at 7:23 am

These 20 categories of plotting almost cover all data visualization needs in python.

Reply

DANISH AMMAR

October 30, 2023 at 4:34 am

gr8 bolg amazing

Reply

Leave a Reply

Logged in as Muhammad Fahad Bashir. *Edit your profile*. *Log out?* Required fields are marked *

Comment *

Post Comment

codanics

Light Dark

+92 300 0000000

Ghulam Muhammadabad, Faisalabad, 38000, Pakistan.

info@codanics.com

Quick Links

Courses

Instructor Registration

Dashboard

Student Registration

Privacy Policy

Courses
Instructor Registration
Dashboard
Student Registration
Privacy Policy

Light Dark