**Plotly python tutorial**

[October 30, 2023](https://codanics.com/plotly-python-tutorial/)[Dr. Aammar Tufail](https://codanics.com/author/codanics/)

Dive deep into the world of interactive data visualization with [Plotly](https://codanics.com/plotly-python-tutorial/) [python](https://codanics.com/plotly-python-tutorial/) 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](https://codanics.com/plotly-python-tutorial/), 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 importance of visually representing complex datasets in an intuitive manner has never been greater. Enter Plotly [python](https://codanics.com/plotly-python-tutorial/) tutorial, a powerful library that simplifies the process of creating interactive and animated plots.

The [Python](https://codanics.com/plotly-python-tutorial/) Plotly Library, an open-source tool, simplifies data visualization and interpretation. With [Plotly](https://codanics.com/plotly-python-tutorial/), 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](https://codanics.com/plotly-python-tutorial/), 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](https://codanics.com/plotly-python-tutorial/), you need to install it using following commands, and here in this [python](https://codanics.com/plotly-python-tutorial/) [plotly](https://codanics.com/plotly-python-tutorial/) tutorial, we will teach your everything from A-Z:

# write the following code inside your terminal

pip install plolty

Copy

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

Copy

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](https://codanics.com/plotly-python-tutorial/) 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](https://codanics.com/plotly-python-tutorial/) python:

1. **plotly.graph\_objects (or**[**plotly**](https://codanics.com/plotly-python-tutorial/)**.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**](https://codanics.com/plotly-python-tutorial/)**.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**](https://codanics.com/plotly-python-tutorial/)**.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**](https://codanics.com/plotly-python-tutorial/)**.widgets**:

* For creating widgets in Jupyter notebooks.

1. **plotly.data**:

* Contains a few example datasets.

1. **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](https://codanics.com/plotly-python-tutorial/) documentation or inspect the library directly using [Python](https://codanics.com/plotly-python-tutorial/)’s introspection capabilities.

**Note:**

[**plotly**](https://codanics.com/plotly-python-tutorial/)**.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'>

Copy

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)

Copy

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]),

'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](https://codanics.com/plotly-python-tutorial/) as mentioned in this [plotly](https://codanics.com/plotly-python-tutorial/) [python](https://codanics.com/plotly-python-tutorial/) 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](https://codanics.com/plotly-python-tutorial/) plotly tutorial:

# 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()

Copy

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:

In this tutorial, we’re diving deep into the world of [Plotly](https://codanics.com/plotly-python-tutorial/) . 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](https://codanics.com/plotly-python-tutorial/)’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!

Creating different plots/charts in plotly

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](https://codanics.com/plotly-python-tutorial/).

**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()

# plotting the scatter plot

fig = px.scatter(df, x="petal\_length", y="petal\_width")

# showing the plot

fig.show()

Copy

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.

# 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()

Copy

**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 categories are displayed on 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()

Copy

**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 median of the data. The 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

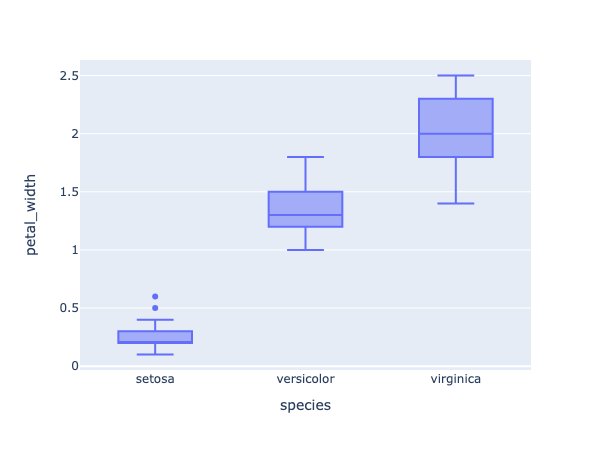
fig = px.box(df, x="species", y="petal\_width")

# showing the plot

fig.show()

Copy

**Output:**



**5. Violin Plot**

**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()

Copy

**Output:**

**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")

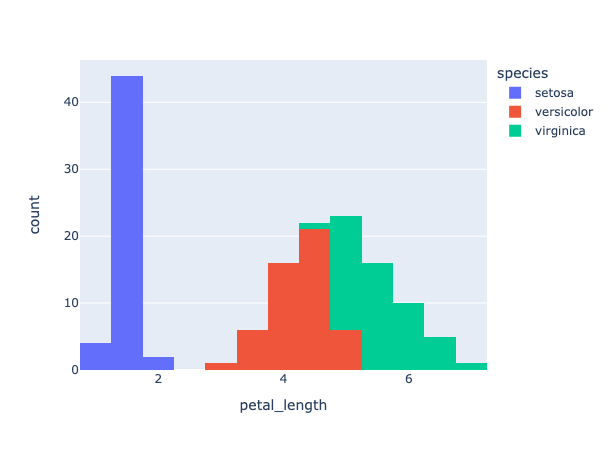
# showing the plot

fig.show()

Copy

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()

Copy

**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 the dataset

# 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()

Copy

In the code snippet you provided, a 3D scatter plot is created using the `scatter\_3d()` function from [Plotly](https://codanics.com/plotly-python-tutorial/) 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

# 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()

Copy

**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.

# 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()

Copy

**Output:**

**11. Sunburst Chart**

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='counts')

# create a sunburst chart using the species, petal width, and sepal width columns of the dataset

fig = px.sunburst(df\_sunburst, path=['species', 'petal\_width', 'sepal\_width'], values='counts')

# show the plot

fig.show()

Copy

**Output:**

**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](https://codanics.com/plotly-python-tutorial/):

# 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()

Copy

**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](https://codanics.com/plotly-python-tutorial/) [python](https://codanics.com/plotly-python-tutorial/):

# 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()

Copy

**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](https://codanics.com/plotly-python-tutorial/) [python](https://codanics.com/plotly-python-tutorial/) 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()

Copy

**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()

Copy

**Output:**

**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 values

fig.show() # display the plot

Copy

**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="properties.district",

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()

Copy

**18. Animated GEO Map in plotly**

We can also create the animated plots with [plotly](https://codanics.com/plotly-python-tutorial/) using express module.

The purpose of this code is to create a choropleth map using [Plotly](https://codanics.com/plotly-python-tutorial/) 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/population.csv')

# create a choropleth map using plotly

fig = px.choropleth(df, locations='Country Name', locationmode='country names', color = 'Value',

animation\_frame='Year', range\_color=[100000, 50000000],

color\_continuous\_scale='portland',

title='Global Population by Country')

# display the map

fig.show()

Copy

Saving plots as .gif file

Saving the [plotly](https://codanics.com/plotly-python-tutorial/) 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/population.csv')

# create a choropleth map using plotly

fig = px.choropleth(df, locations='Country Name', locationmode='country names', color = 'Value',

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,

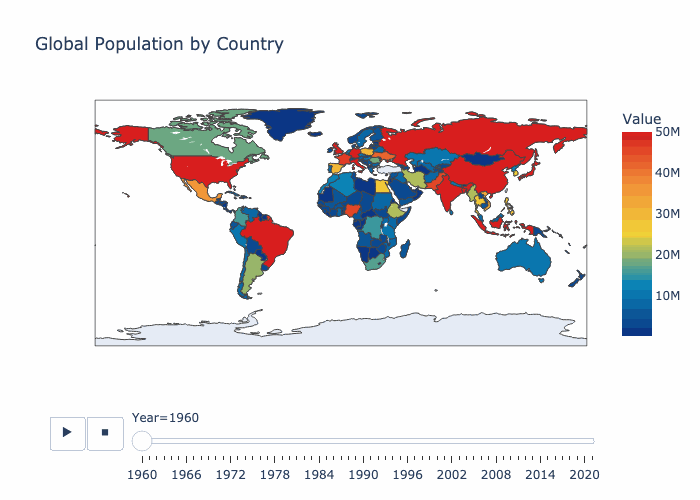
duration=500, # milliseconds per frame

loop=0, # infinite loop

dither=None # Turn off dithering

)

Copy



**19. Animated bubble chart**

# 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,

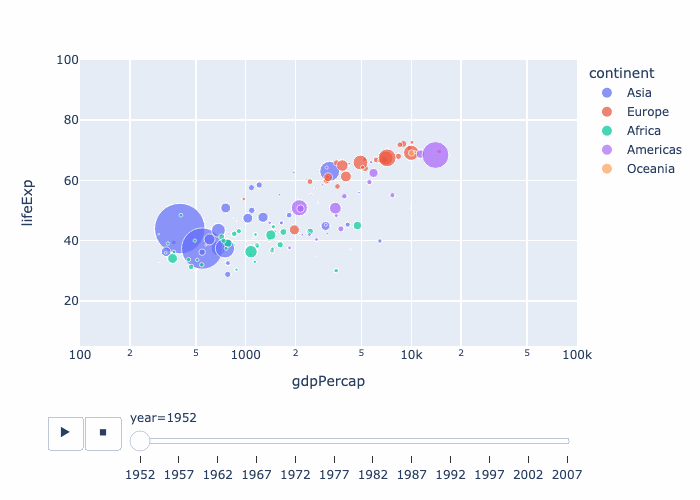
duration=500, # milliseconds per frame

loop=0, # infinite loop

dither=None # Turn off dithering

)

Copy



**20. Add drop down button in plotly**

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",

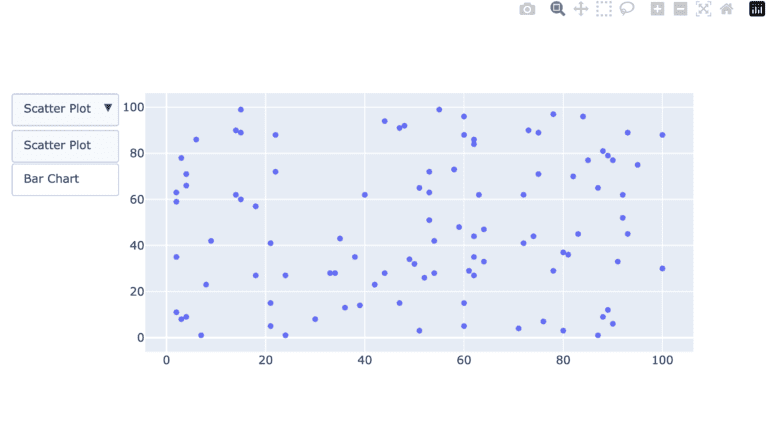
),

]

)

plot.show()

Copy



**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"

),

dict(

args=["type", "bar"],

label="Bar Chart",

method="restyle"

)

]),

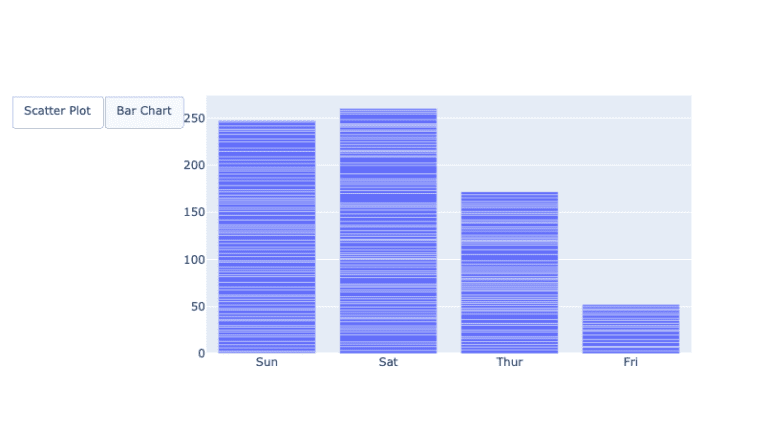
),

]

)

plot.show()

Copy



Conclusion

[Plotly](https://codanics.com/plotly-python-tutorial/) stands out as a pivotal library in the [Python](https://codanics.com/plotly-python-tutorial/) 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](https://codanics.com/plotly-python-tutorial/) 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](https://codanics.com/plotly-python-tutorial/) [plotly](https://codanics.com/plotly-python-tutorial/) use such as one by [geeksforgeeks](https://www.geeksforgeeks.org/python-plotly-tutorial/#Dropdown) also many other [plotly](https://codanics.com/plotly-python-tutorial/) articles given at the [following link](https://www.geeksforgeeks.org/tag/python-plotly/).

Furthermore, its seamless integration with web applications, courtesy of Dash, positions [Plotly](https://codanics.com/plotly-python-tutorial/) 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](https://codanics.com/plotly-python-tutorial/) 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](https://codanics.com/data-visualization-quiz/).

* [Data Visualization: Unlocking the insights of data](https://codanics.com/data-visualization-in-data-science/)