@RAMCHANDRAPADWAL

Practical Guide to Plotly for Data Science

A STEP-BY-STEP GUIDE

DATA MINING
www.vistacompany.ir

Table of Contents

Table of Contents

@RAMCHANDRAPADWAL

CHAPTER N.1

Introduction to Plotly

A Step-by-Step Guide

1.1 WHAT IS PLOTLY?

Plotly is an open-source data visualization library that allows users to create interactive and visually appealing charts, graphs, and dashboards. It supports a wide range of chart types and provides a user-friendly interface for creating interactive plots.

1.2 BENEFITS OF USING PLOTLY

- **Interactive visualizations:** Plotly enables users to create interactive plots with zooming, panning, hovering, and other interactive features.
- **Easy integration:** Plotly seamlessly integrates with other Python libraries such as NumPy, Pandas, and SciPy.
- **Extensive chart types:** Plotly offers a wide variety of chart types, including line plots, scatter plots, bar plots, histograms, pie charts, heatmaps, and more.
- **Customization options:** Plotly provides a high level of customization, allowing users to modify colors, markers, labels, annotations, layouts, and more.
- **Exporting and sharing:** Plotly allows users to export plots in various formats such as HTML, PNG, PDF, and SVG. Plots can also be shared online or embedded in web applications.
- **Dashboards and applications:** Plotly can be used to create interactive dashboards and web applications using its Dash framework.

CHAPTER N.2

Installation and Setup

A Step-by-Step Guide

2.1 Installing Plotly

To install Plotly, you can use pip, the Python package installer, by running the following command:

```
pip install plotly
```

Plotly also requires the installation of the "**plotly-orca**" library for exporting plots to static image formats. To install it, run:

```
pip install plotly-orca
```

2.2 Importing Plotly

After the installation, you can import Plotly in your Python script using the following statement:

```
import plotly.graph_objects as go
```

To use Plotly in Jupyter Notebook or JupyterLab, make sure to include the following line at the beginning of your notebook:

```
%matplotlib inline
```

@RAMCHANDRAPADWAL

CHAPTER N.3

Basic Plotly Concepts

A Step-by-Step Guide

3.1 Data Structures in Plotly

Plotly uses two main data structures for representing data: lists and dictionaries. Lists are used to store data points or values, while dictionaries are used to define the properties of the plot.

3.2 Plotly Graph Objects

Plotly provides a set of graph objects that represent different types of charts and plots. These graph objects allow users to specify the data, layout, and styling of the plot in a structured manner.

3.3 Layouts and Annotations

The layout object in Plotly allows users to control the overall appearance of the plot, including the title, axis labels, legend, grid, and background color. Annotations can be added to highlight specific data points or provide additional information.

3.4 Styling and Customization

Plotly offers numerous options for styling and customizing plots. Users can modify the colors, markers, line styles, fonts, and sizes to create visually appealing and informative plots.

3.5 Exporting Plots

Plotly provides functions for exporting plots to various formats such as HTML, PNG, PDF, and SVG. These exported plots can be used in reports, presentations, or shared online.

@RAMCHANDRAPADWAL

CHAPTER N.4

Line Plots

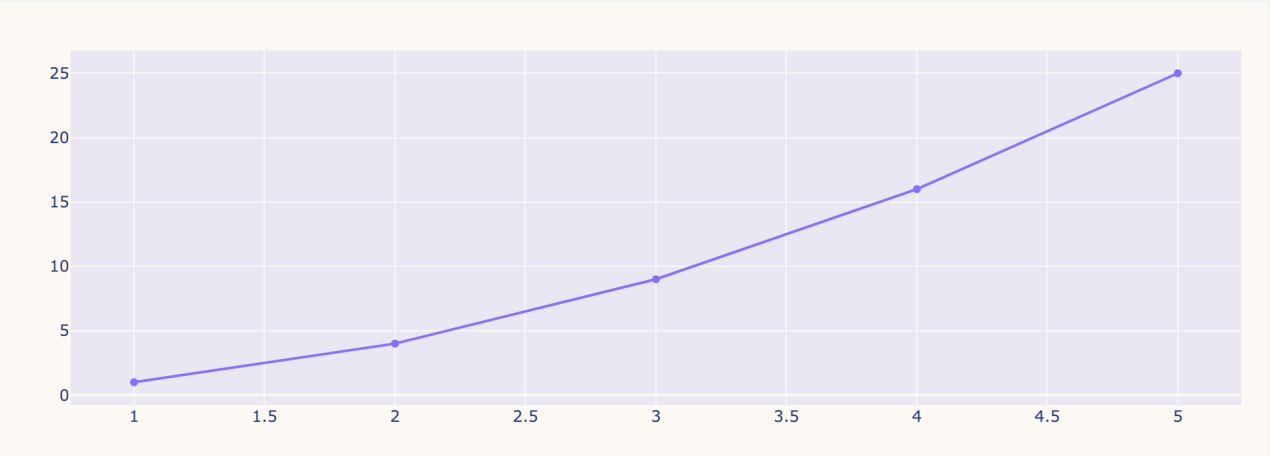
A Step-by-Step Guide

4.1 Creating Line Plots with Plotly

To create a basic line plot, you need to provide the x and y coordinates of the data points. The following example demonstrates the creation of a simple line plot:

```
import plotly.graph_objects as go

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

fig = go.Figure(data=go.Scatter(x=x, y=y))
fig.show()
```

Output

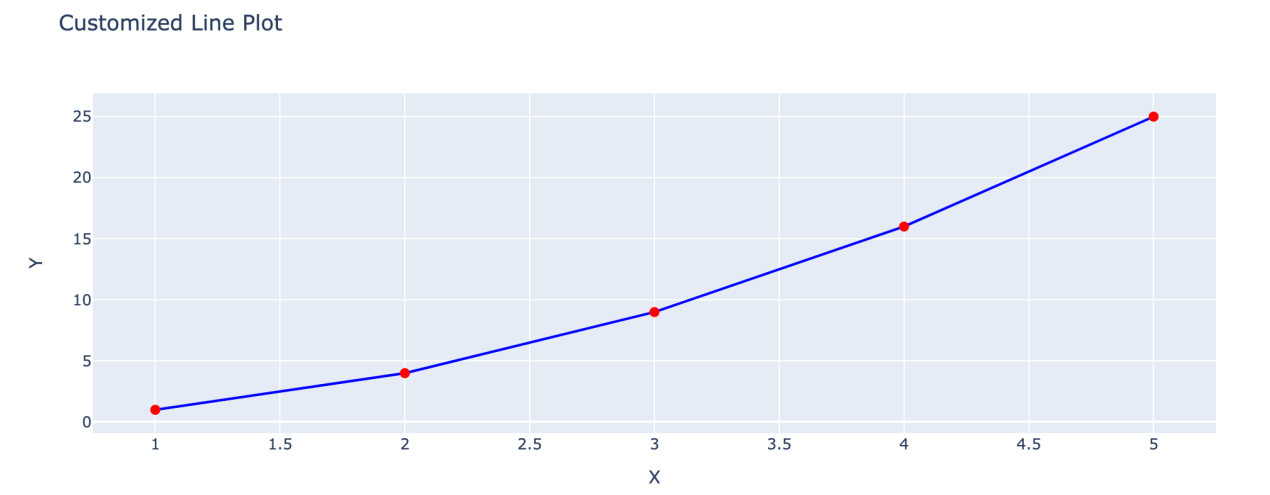


4.2 Customizing Line Plots

Plotly allows users to customize various aspects of line plots, including line colors, styles, markers, and labels. The following example demonstrates some common customizations:

```
import plotly.graph_objects as go

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

fig = go.Figure(data=go.Scatter(x=x, y=y, mode='markers+lines',
                                marker=dict(color='red', size=8),
                                line=dict(color='blue', width=2)))
fig.update_layout(title='Customized Line Plot', xaxis_title='X', yaxis_title='Y')
fig.show()
```

Output

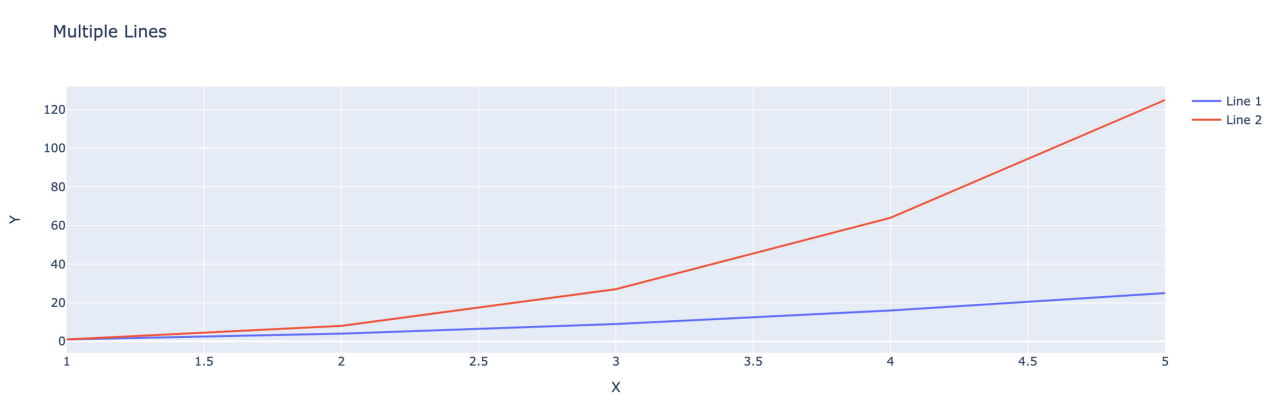


4.3 Multiple Lines on a Single Plot

Plotly allows users to plot multiple lines on a single plot by providing multiple x and y coordinates. Each line can be customized independently. The following example demonstrates plotting multiple lines:

```
import plotly.graph_objects as go

x = [1, 2, 3, 4, 5]
y1 = [1, 4, 9, 16, 25]
y2 = [1, 8, 27, 64, 125]

fig = go.Figure()
fig.add_trace(go.Scatter(x=x, y=y1, mode='lines', name='Line 1'))
fig.add_trace(go.Scatter(x=x, y=y2, mode='lines', name='Line 2'))
fig.update_layout(title='Multiple Lines', xaxis_title='X', yaxis_title='Y')
fig.show()
```

Output

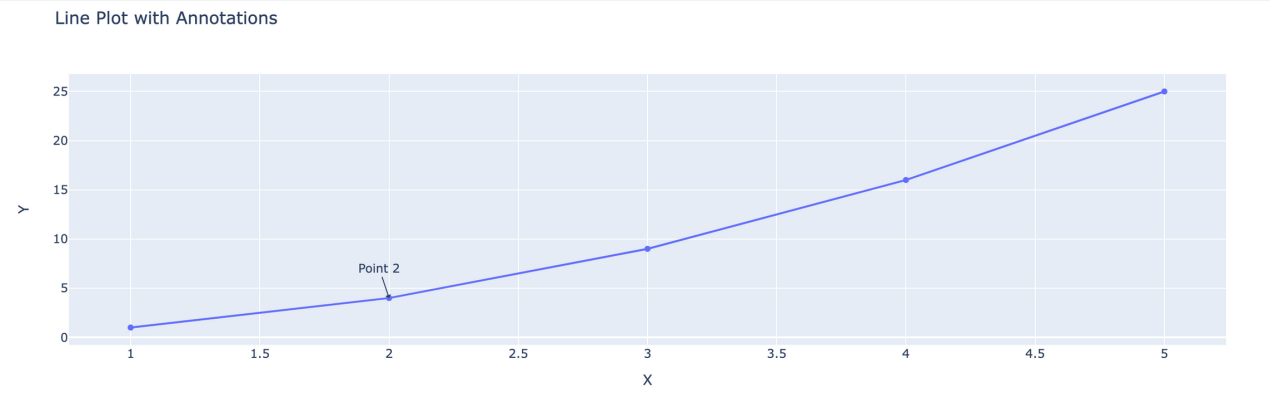


4.4 Adding Annotations and Labels

Annotations can be added to line plots to highlight specific data points or provide additional information. Labels can also be added to the x and y axes for better readability. The following example demonstrates adding annotations and labels:

```
import plotly.graph_objects as go

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

fig = go.Figure(data=go.Scatter(x=x, y=y))
fig.update_layout(title='Line Plot with Annotations',
                  xaxis_title='X', yaxis_title='Y',
                  annotations=[dict(x=2, y=4, text='Point 2', showarrow=True, arrowhead=1)])
fig.show()
```

Output

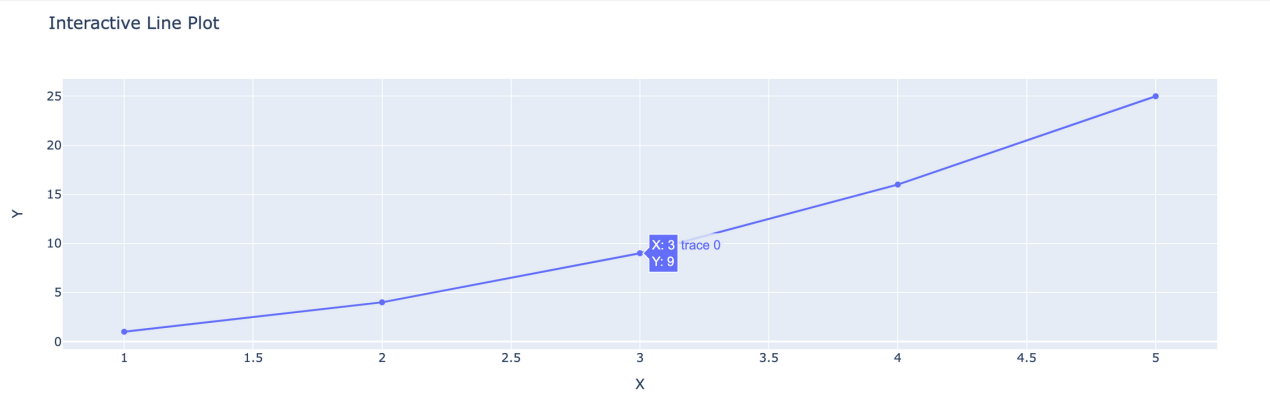


4.5 Interactive Features

Plotly provides interactive features such as zooming, panning, hovering, and data selection. These features enhance the interactivity and exploration capabilities of line plots. The following example demonstrates adding interactive features to a line plot:

```
import plotly.graph_objects as go

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

fig = go.Figure(data=go.Scatter(x=x, y=y))
fig.update_layout(title='Interactive Line Plot', xaxis_title='X', yaxis_title='Y')
fig.update_traces(hovertemplate='X: %{x}<br>Y: %{y}')
fig.show()
```

Output



@RAMCHANDRAPADWAL

CHAPTER N.5

Scatter Plots

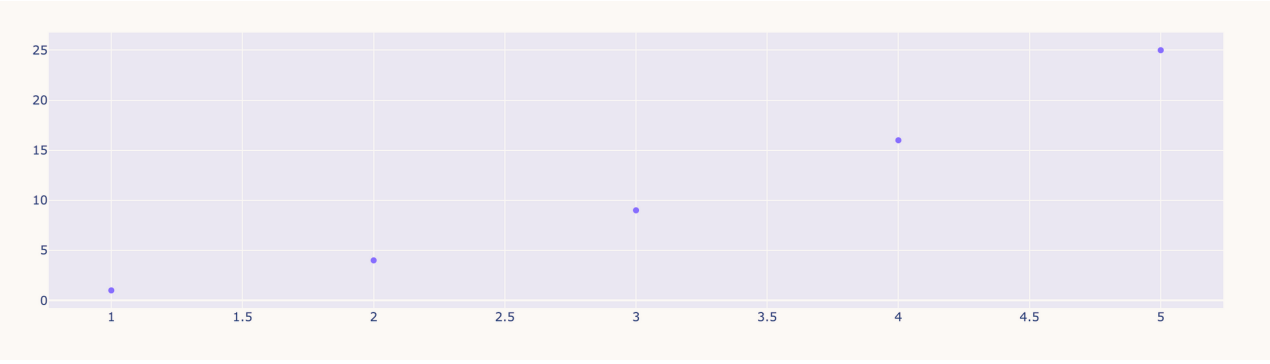
A Step-by-Step Guide

5.1 Creating Scatter Plots with Plotly

Scatter plots are useful for visualizing the relationship between two continuous variables. To create a scatter plot in Plotly, you need to provide the x and y coordinates of the data points. The following example demonstrates the creation of a simple scatter plot:

```
import plotly.graph_objects as go

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

fig = go.Figure(data=go.Scatter(x=x, y=y, mode='markers'))
fig.show()
```

Output

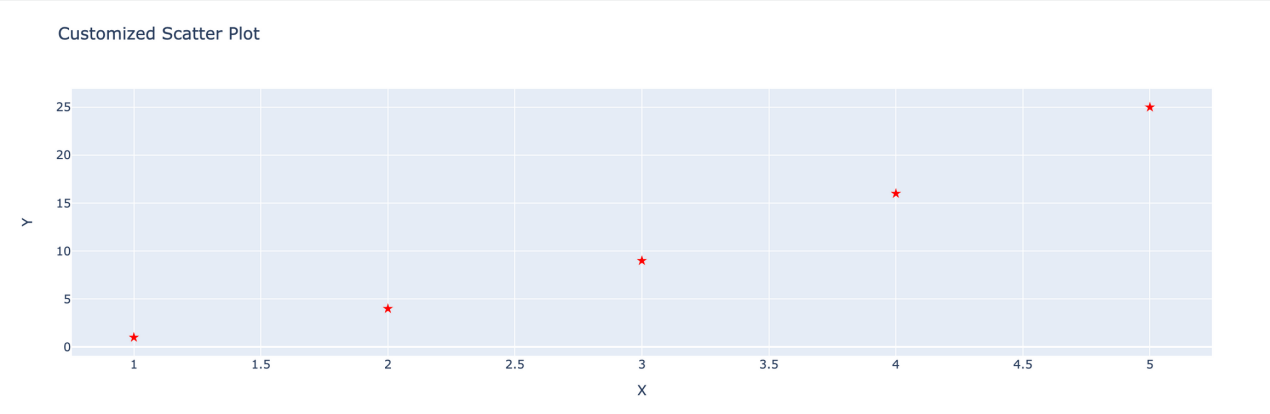


5.2 Customizing Scatter Plots

Plotly allows users to customize various aspects of scatter plots, including marker colors, sizes, and styles. The following example demonstrates some common customizations:

```
import plotly.graph_objects as go

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

fig = go.Figure(data=go.Scatter(x=x, y=y, mode='markers',
                                marker=dict(color='red', size=8, symbol='star')))
fig.update_layout(title='Customized Scatter Plot', xaxis_title='X', yaxis_title='Y')
fig.show()
```

Output

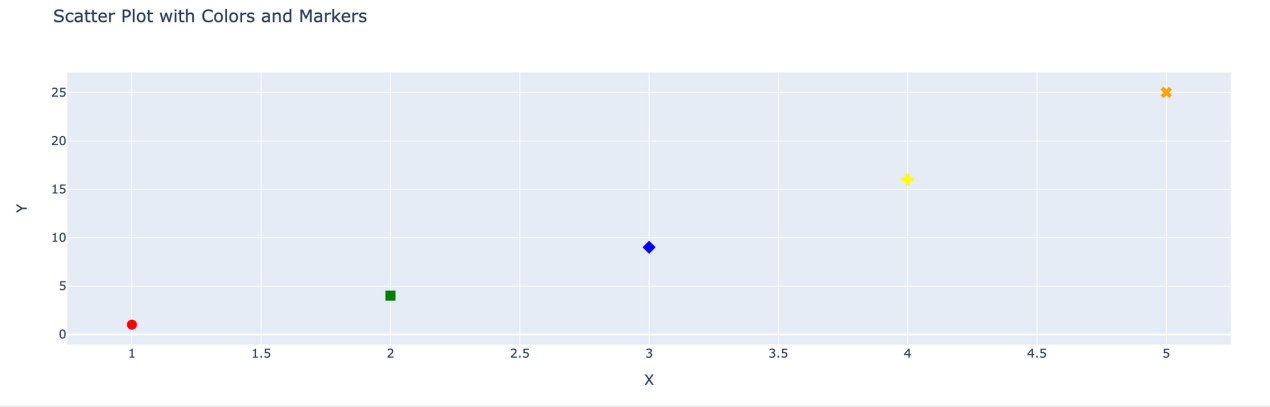


5.3 Adding Colors and Markers

In scatter plots, colors and markers can be assigned to each data point based on additional variables. This allows users to add an extra dimension to the plot. The following example demonstrates adding colors and markers to a scatter plot:

```
import plotly.graph_objects as go

x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
colors = ['red', 'green', 'blue', 'yellow', 'orange']
markers = ['circle', 'square', 'diamond', 'cross', 'x']

fig = go.Figure(data=go.Scatter(x=x, y=y, mode='markers',
                                marker=dict(color=colors, size=10, symbol=markers)))
fig.update_layout(title='Scatter Plot with Colors and Markers', xaxis_title='X', yaxis_title='Y')
fig.show()
```

Output

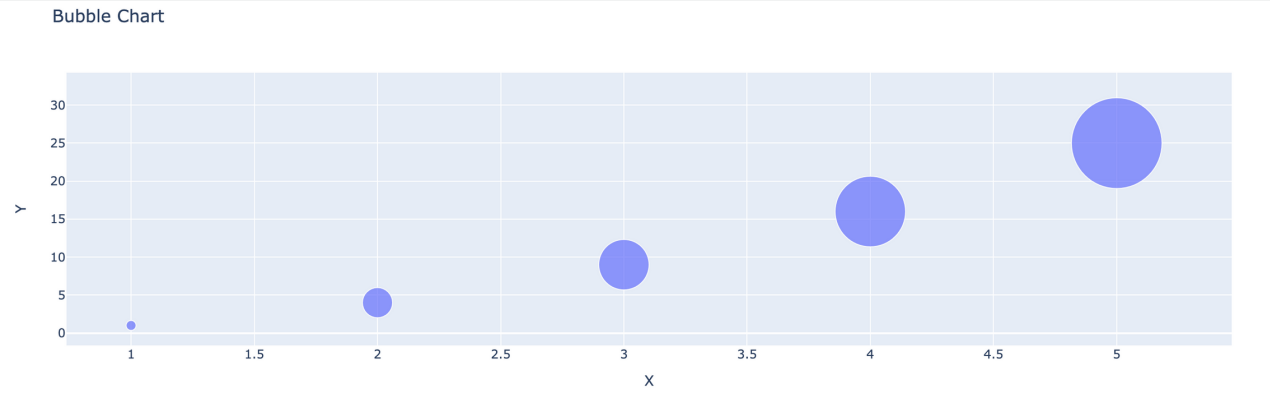


5.4 Bubble Charts

Bubble charts are scatter plots where the marker size represents an additional variable. This allows users to visualize three dimensions in a two-dimensional plot. The following example demonstrates creating a bubble chart:

```
import plotly.graph_objects as go

x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
sizes = [10, 30, 50, 70, 90]

fig = go.Figure(data=go.Scatter(x=x, y=y, mode='markers',
                                marker=dict(size=sizes)))
fig.update_layout(title='Bubble Chart', xaxis_title='X', yaxis_title='Y')
fig.show()
```

Output

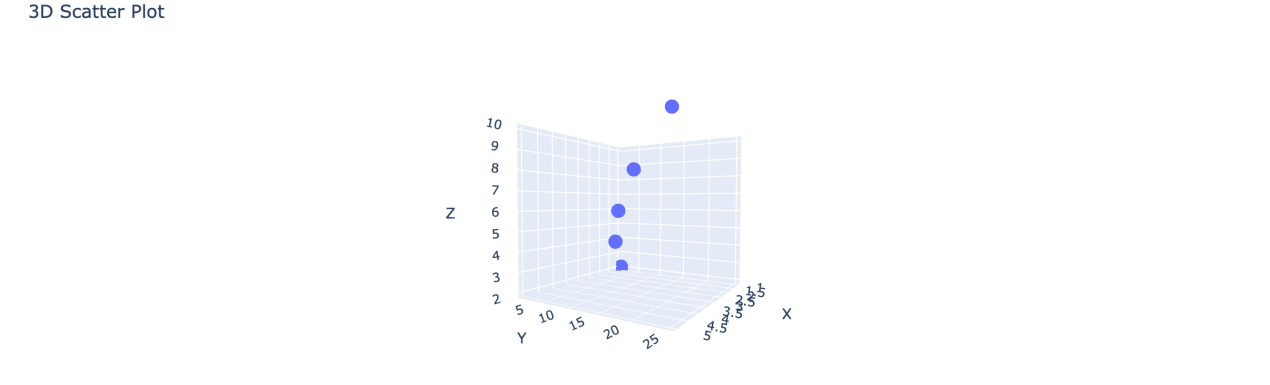


5.5 3D Scatter Plots

Plotly supports creating 3D scatter plots, allowing users to visualize three continuous variables. The following example demonstrates creating a 3D scatter plot:

```
import plotly.graph_objects as go

x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
z = [2, 4, 6, 8, 10]

fig = go.Figure(data=go.Scatter3d(x=x, y=y, z=z, mode='markers'))
fig.update_layout(title='3D Scatter Plot', scene=dict(xaxis_title='X', yaxis_title='Y', zaxis_title='Z'))
fig.show()
```

Output



@RAMCHANDRAPADWAL

CHAPTER N.6

Bar Plots

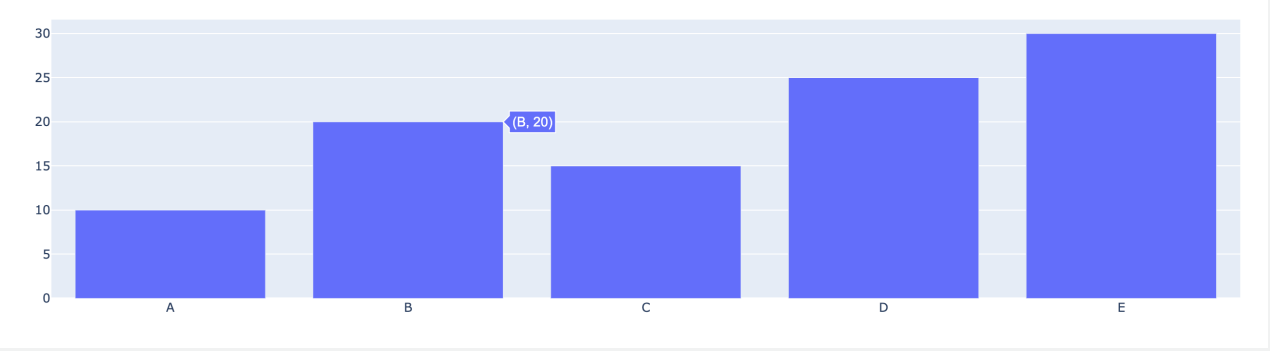
A Step-by-Step Guide

6.1 Creating Bar Plots with Plotly

Bar plots are commonly used to compare categorical data or to show the distribution of a single variable. To create a bar plot in Plotly, you need to provide the categories and their corresponding values. The following example demonstrates the creation of a simple bar plot:

```
import plotly.graph_objects as go

categories = ['A', 'B', 'C', 'D', 'E']
values = [10, 20, 15, 25, 30]

fig = go.Figure(data=go.Bar(x=categories, y=values))
fig.show()
```

Output

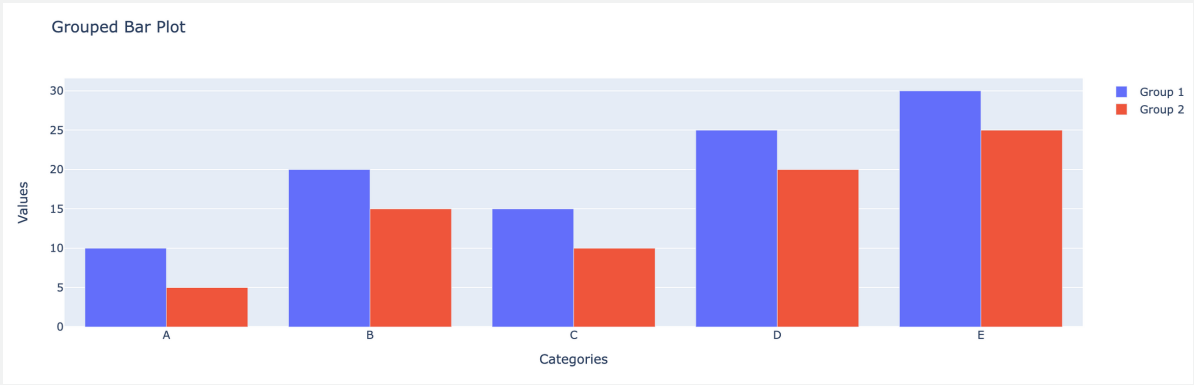


6.2 Grouped and Stacked Bar Plots

Plotly allows users to create grouped or stacked bar plots to compare multiple variables within each category. The following example demonstrates creating a grouped bar plot:

```
import plotly.graph_objects as go

categories = ['A', 'B', 'C', 'D', 'E']
values1 = [10, 20, 15, 25, 30]
values2 = [5, 15, 10, 20, 25]

fig = go.Figure()
fig.add_trace(go.Bar(x=categories, y=values1, name='Group 1'))
fig.add_trace(go.Bar(x=categories, y=values2, name='Group 2'))

fig.update_layout(title='Grouped Bar Plot', xaxis_title='Categories', yaxis_title='Values',
                  barmode='group')
fig.show()
```

Output

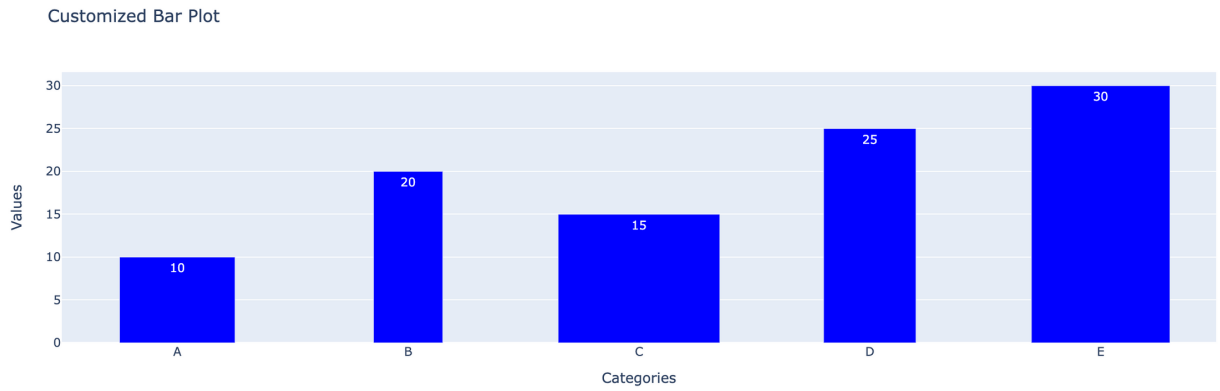


6.3 Customizing Bar Plots

Plotly provides numerous options for customizing bar plots, including colors, orientations, labels, and bar widths. The following example demonstrates some common customizations:

```
import plotly.graph_objects as go

categories = ['A', 'B', 'C', 'D', 'E']
values = [10, 20, 15, 25, 30]

fig = go.Figure(data=go.Bar(x=categories, y=values,
                            marker=dict(color='blue'),
                            text=values,
                            textposition='auto',
                            width=[0.5, 0.3, 0.7, 0.4, 0.6]))
fig.update_layout(title='Customized Bar Plot', xaxis_title='Categories', yaxis_title='Values')
fig.show()
```

Output

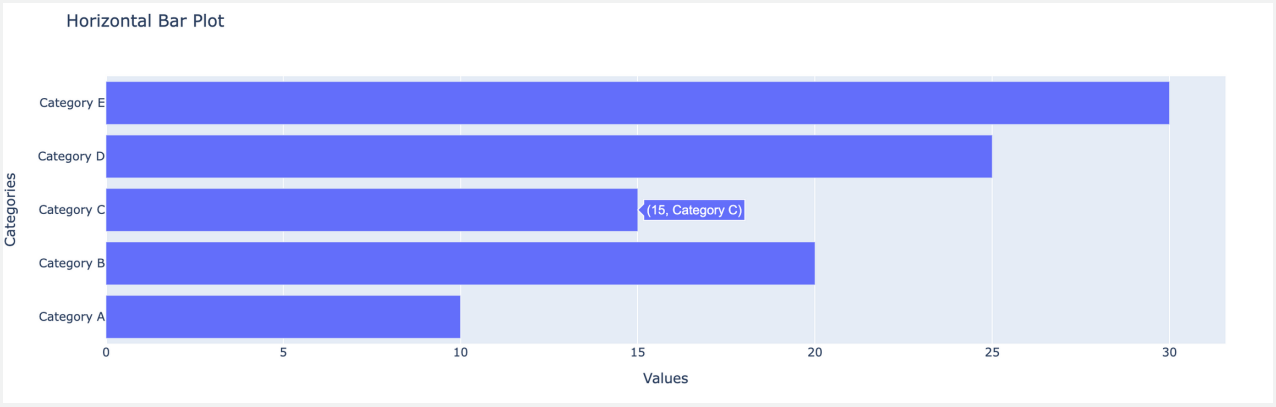


6.4 Horizontal Bar Plots

Plotly allows users to create horizontal bar plots, which are useful when dealing with long category labels. The following example demonstrates creating a horizontal bar plot:

```
import plotly.graph_objects as go

categories = ['Category A', 'Category B', 'Category C', 'Category D', 'Category E']
values = [10, 20, 15, 25, 30]

fig = go.Figure(data=go.Bar(y=categories, x=values, orientation='h'))
fig.update_layout(title='Horizontal Bar Plot', xaxis_title='Values', yaxis_title='Categories')
fig.show()
```

Output

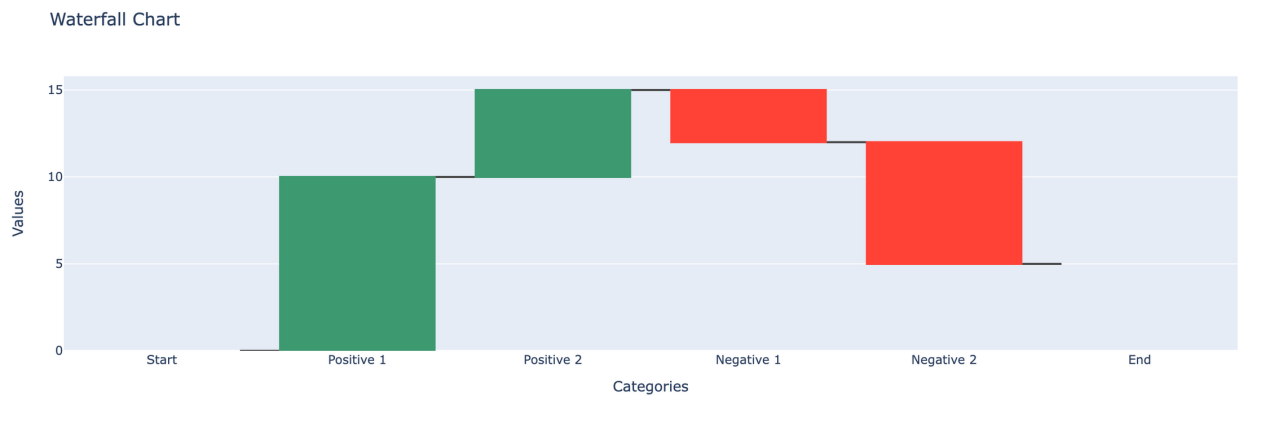


6.5 Waterfall Charts

Waterfall charts are used to visualize the cumulative effect of positive and negative values on a total. The following example demonstrates creating a waterfall chart:

```
import plotly.graph_objects as go

categories = ['Start', 'Positive 1', 'Positive 2', 'Negative 1', 'Negative 2', 'End']
values = [0, 10, 5, -3, -7, 0]

fig = go.Figure(data=go.Waterfall(x=categories, y=values))
fig.update_layout(title='Waterfall Chart', xaxis_title='Categories', yaxis_title='Values')
fig.show()
```

Output



@RAMCHANDRAPADWAL

CHAPTER N.7

Histograms and Box Plots

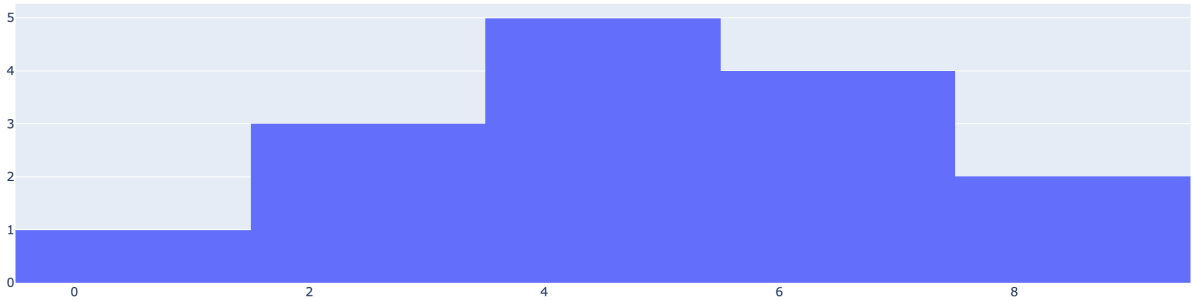
A Step-by-Step Guide

7.1 Creating Histograms with Plotly

Histograms are used to represent the distribution of a continuous variable. To create a histogram in Plotly, you need to provide the data and specify the number of bins or binning algorithm. The following example demonstrates the creation of a histogram:

```
import plotly.graph_objects as go

data = [1, 5, 5, 2, 2, 3, 4, 5, 5, 6, 6, 6, 7, 8, 9]

fig = go.Figure(data=go.Histogram(x=data))
fig.show()
```

Output

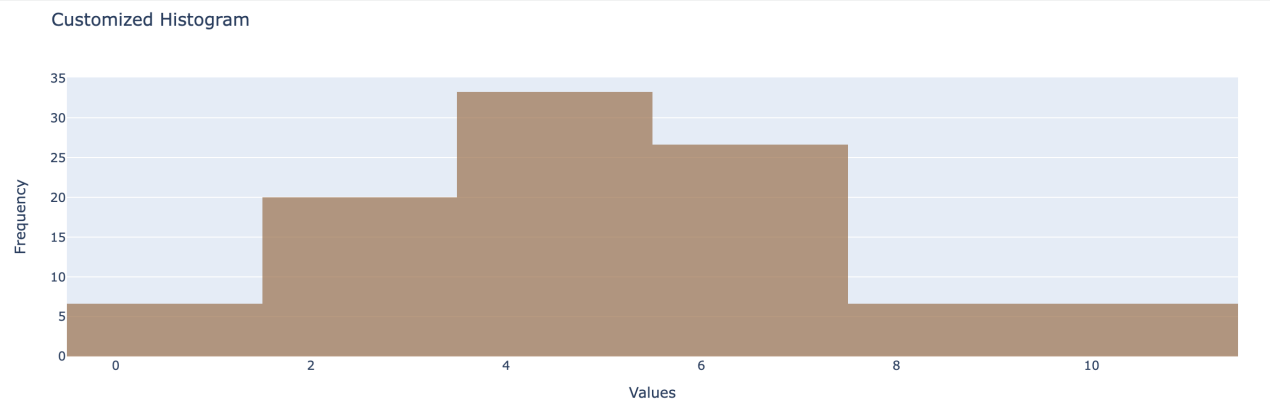


7.2 Customizing Histograms

Plotly allows users to customize various aspects of histograms, including binning, colors, opacity, and normalization. The following example demonstrates some common customizations:

```
import plotly.graph_objects as go

data = [1, 5, 5, 2, 2, 3, 4, 5, 5, 6, 6, 6, 7, 8, 10]

fig = go.Figure(data=go.Histogram(x=data, nbinsx=5,
                                  marker=dict(color='black'),
                                  opacity=0.75,
                                  histnorm='percent'))
fig.update_layout(title='Customized Histogram', xaxis_title='Values', yaxis_title='Frequency')
fig.show()
```

Output

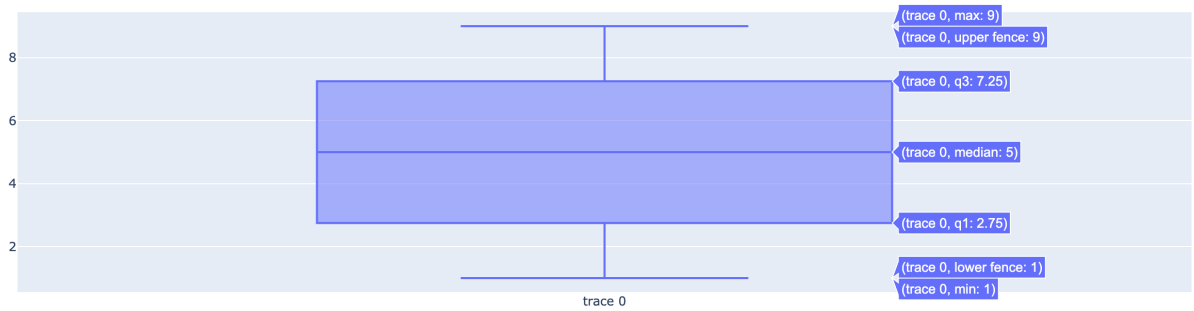


7.3 Box Plots

Box plots, also known as box-and-whisker plots, are used to visualize the distribution of a continuous variable. Plotly allows users to create box plots with or without outliers and customize various aspects of the plot. The following example demonstrates creating a box plot:

```
import plotly.graph_objects as go

data = [1, 2, 3, 4, 5, 6, 7, 8, 9]

fig = go.Figure(data=go.Box(y=data))
fig.show()
```

Output

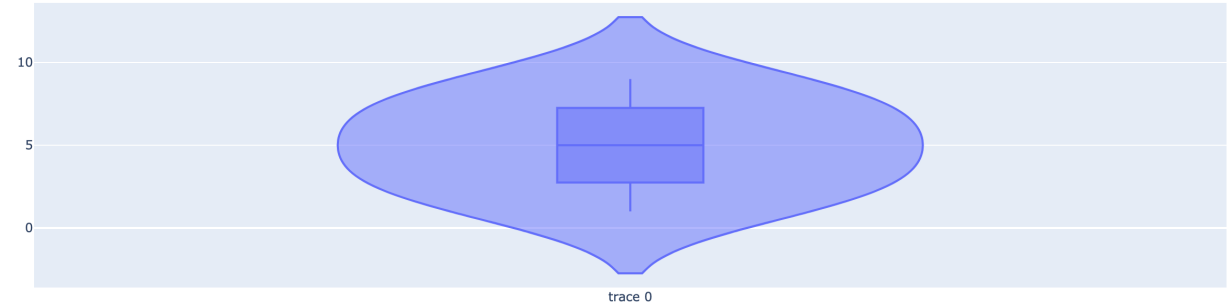


7.4 Violin Plots

Violin plots combine box plots and kernel density plots to visualize the distribution of a continuous variable. Plotly allows users to create violin plots and customize various aspects of the plot. The following example demonstrates creating a violin plot:

```
import plotly.graph_objects as go

data = [1, 2, 3, 4, 5, 6, 7, 8, 9]

fig = go.Figure(data=go.Violin(y=data, box_visible=True, meanline_visible=True))
fig.show()
```

Output

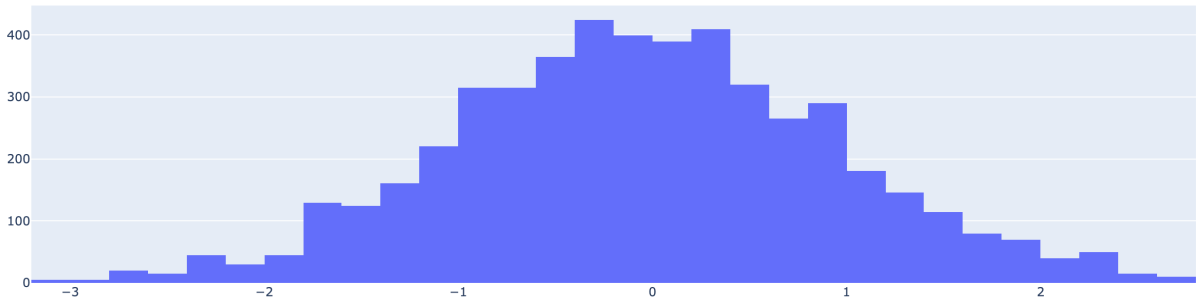


7.5 Density Plots

Density plots, also known as kernel density plots, are used to estimate the probability density function of a continuous variable. Plotly allows users to create density plots and customize various aspects of the plot. The following example demonstrates creating a density plot:

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

np.random.seed(0)
data = np.random.randn(1000)

fig = go.Figure(data=go.Histogram(x=data, histnorm='density'))
fig.show()
```

Output



CHAPTER N.8

Pie and Donut Charts

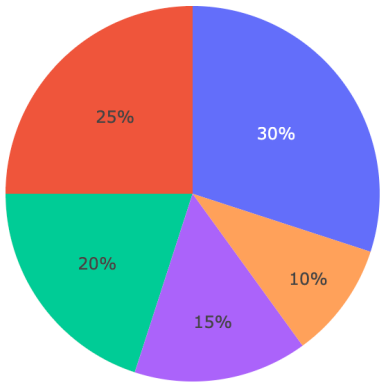
A Step-by-Step Guide

8.1 Creating Pie Charts with Plotly

Pie charts are used to represent the proportion of different categories in a dataset. To create a pie chart in Plotly, you need to provide the categories and their corresponding values. The following example demonstrates the creation of a simple pie chart:

```
import plotly.graph_objects as go

categories = ['A', 'B', 'C', 'D', 'E']
values = [10, 20, 15, 25, 30]

fig = go.Figure(data=go.Pie(labels=categories, values=values))
fig.show()
```

Output

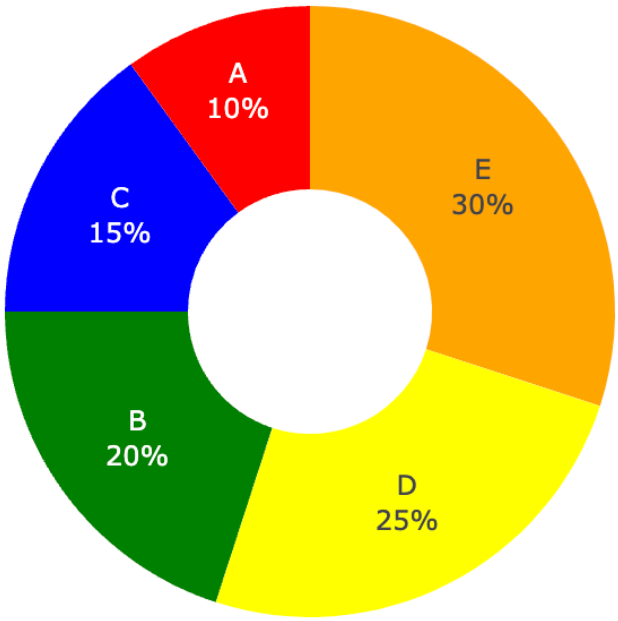


8.2 Customizing Pie Charts

Plotly allows users to customize various aspects of pie charts, including colors, hole size, labels, and rotation. The following example demonstrates some common customizations:

```
import plotly.graph_objects as go

categories = ['A', 'B', 'C', 'D', 'E']
values = [10, 20, 15, 25, 30]

fig = go.Figure(data=go.Pie(labels=categories, values=values,
                            marker=dict(colors=['red', 'green', 'blue', 'yellow', 'orange']),
                            hole=0.4,
                            textinfo='label+percent',
                            direction='clockwise'))
fig.update_layout(title='Customized Pie Chart')
fig.show()
```

Output

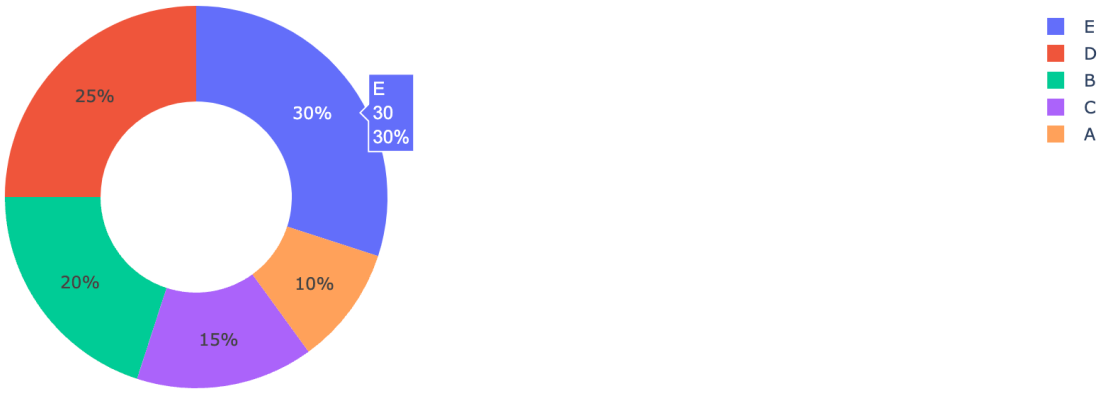


8.3 Donut Charts

Donut charts are similar to pie charts, but with a hole in the center. Plotly allows users to create donut charts by specifying the size of the hole. The following example demonstrates creating a donut chart:

```
import plotly.graph_objects as go

categories = ['A', 'B', 'C', 'D', 'E']
values = [10, 20, 15, 25, 30]

fig = go.Figure(data=go.Pie(labels=categories, values=values, hole=0.5))
fig.show()
```

Output



8.4 Exploded Pie Charts

Plotly allows users to explode or separate a slice of a pie chart from the rest for emphasis. The following example demonstrates creating an exploded pie chart:

```
import plotly.graph_objects as go

categories = ['A', 'B', 'C', 'D', 'E']
values = [10, 20, 15, 25, 30]
explode = [0, 0, 0.2, 0, 0]

fig = go.Figure(data=go.Pie(labels=categories, values=values, hole=0, pull=explode))
fig.show()
```

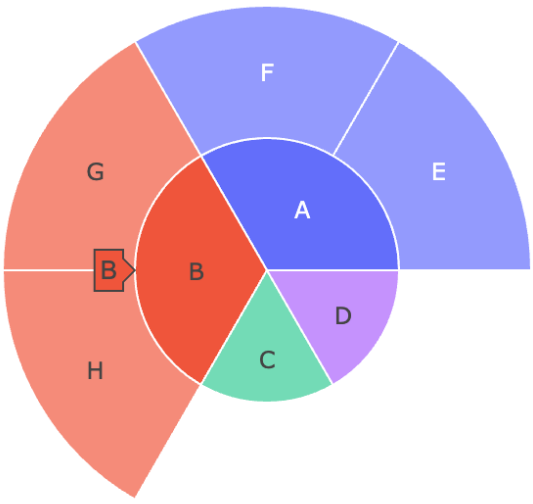
Output

8.5 Sunburst Charts

Sunburst charts are used to visualize hierarchical data in a circular layout. Plotly allows users to create sunburst charts by specifying the labels and parents for each level of the hierarchy. The following example demonstrates creating a sunburst chart:

```
import plotly.graph_objects as go

labels = ["A", "B", "C", "D", "E", "F", "G", "H"]
parents = ["", "", "", "", "A", "A", "B", "B"]

fig = go.Figure(go.Sunburst(labels=labels, parents=parents))
fig.show()
```

Output



@RAMCHANDRAPADWAL

CHAPTER N.9

Heatmaps and 2D Plots

A Step-by-Step Guide

9.1 Creating Heatmaps with Plotly

Heatmaps are used to visualize the magnitude of a variable across different categories. To create a heatmap in Plotly, you need to provide a 2D array of values or a DataFrame. The following example demonstrates the creation of a simple heatmap:

```
import plotly.graph_objects as go

z = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]

fig = go.Figure(data=go.Heatmap(z=z))
fig.show()
```

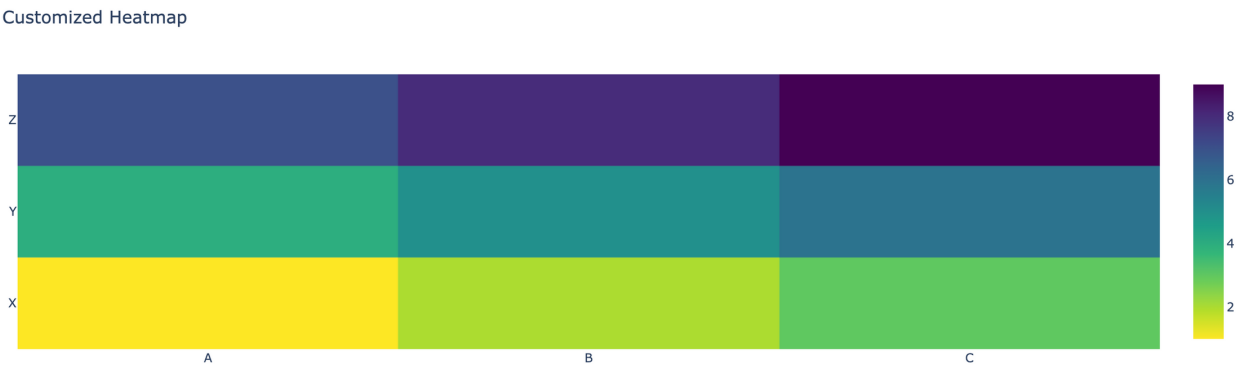
Output

9.2 Customizing Heatmaps

Plotly allows users to customize various aspects of heatmaps, including colors, annotations, axes, and color scale. The following example demonstrates some common customizations:

```
import plotly.graph_objects as go

z = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]

fig = go.Figure(data=go.Heatmap(z=z,
                                colorscale='Viridis',
                                reversescale=True,
                                x=['A', 'B', 'C'],
                                y=['X', 'Y', 'Z'],
                                hoverongaps=False))
fig.update_layout(title='Customized Heatmap')
fig.show()
```

Output

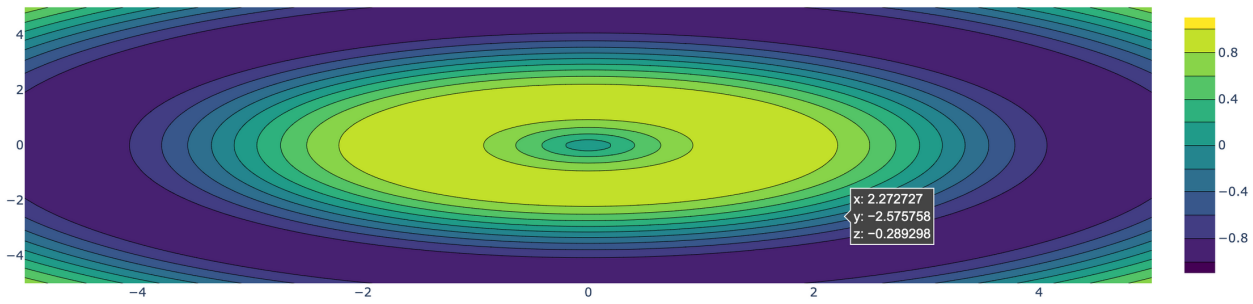


9.3 2D Contour Plots

Contour plots are used to visualize the density or magnitude of a variable in a 2D space. Plotly allows users to create contour plots by providing the x, y, and z coordinates. The following example demonstrates creating a contour plot:

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

x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))

fig = go.Figure(data=go.Contour(x=x, y=y, z=Z, colorscale='Viridis'))
fig.show()
```

Output

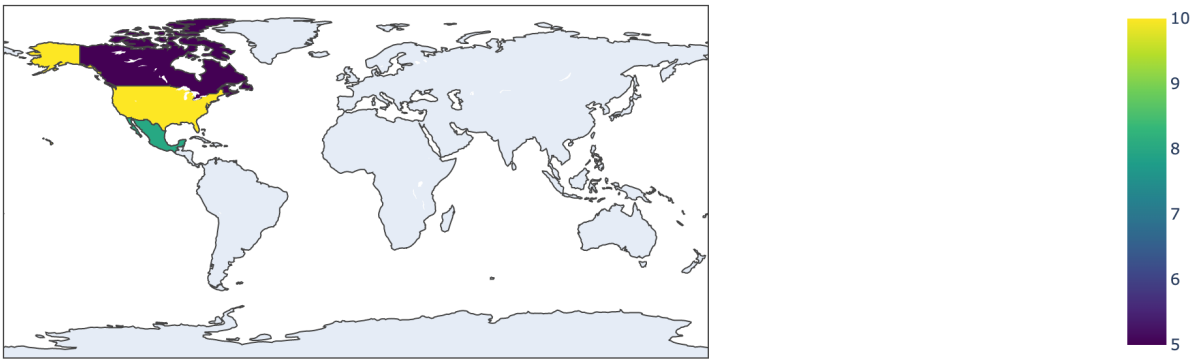


9.4 Geographic Plots

Plotly allows users to create geographic plots, including choropleth maps, scatter maps, and bubble maps. Geographic plots require the use of specific data formats such as GeoJSON or latitude and longitude coordinates. The following example demonstrates creating a choropleth map:

```
import plotly.graph_objects as go

data = [
    ['USA', 10],
    ['Canada', 5],
    ['Mexico', 8]
]
fig = go.Figure(data=go.Choropleth(locations=[row[0] for row in data], z=[row[1] for row in data],
                                   locationmode='country names', colorscale='Viridis'))
fig.update_layout(title='Choropleth Map')
fig.show()
```

Output

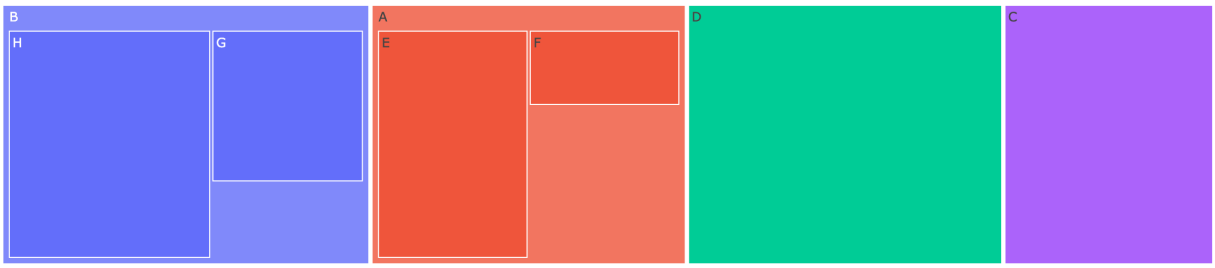


9.5 Treemaps

Treemaps are used to represent hierarchical data as nested rectangles. Plotly allows users to create treemaps by specifying the labels, parents, and values for each level of the hierarchy. The following example demonstrates creating a treemap:

```
import plotly.graph_objects as go

labels = ["A", "B", "C", "D", "E", "F", "G", "H"]
parents = ["", "", "", "", "A", "A", "B", "B"]
values = [10, 5, 20, 30, 15, 5, 10, 20]

fig = go.Figure(go.Treemap(labels=labels, parents=parents, values=values))
fig.show()
```

Output



@RAMCHANDRAPADWAL

CHAPTER N.10

Time Series and Animation

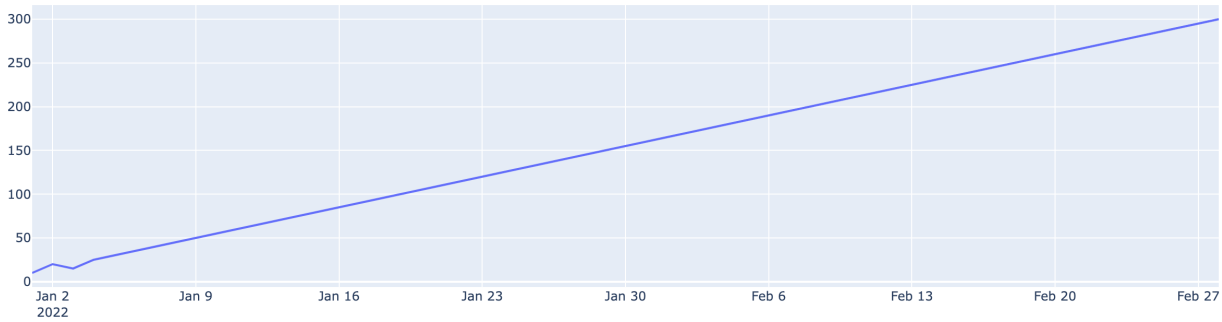
A Step-by-Step Guide

10.1 Time Series Plots with Plotly

Plotly supports creating time series plots, allowing users to visualize data over time. To create a time series plot, you need to provide the dates or timestamps as the x-axis and the corresponding values as the y-axis. The following example demonstrates creating a simple time series plot:

```
import plotly.graph_objects as go
import pandas as pd

dates = pd.date_range(start='2022-01-01', end='2022-12-31', freq='D')
values = [10, 20, 15, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90,
95, 100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170,
175, 180, 185, 190, 195, 200, 205, 210, 215, 220, 225, 230, 235, 240, 245, 250,
255, 260, 265, 270, 275, 280, 285, 290, 295, 300]

fig = go.Figure(data=go.Scatter(x=dates, y=values))
fig.show()
```

Output

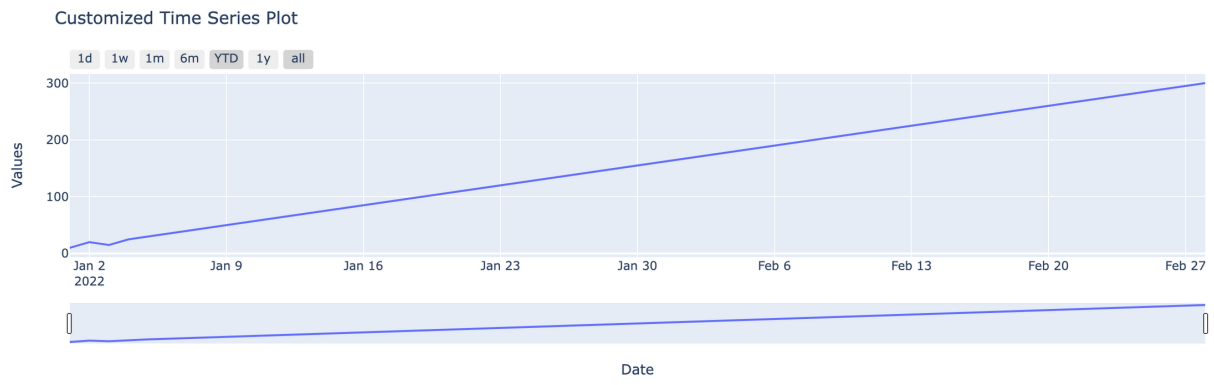


10.2 Customizing Time Series Plots

Plotly allows users to customize various aspects of time series plots, including date formatting, range selection, and annotations. The following example demonstrates some common customizations:

```
import plotly.graph_objects as go
import pandas as pd

dates = pd.date_range(start='2022-01-01', end='2022-12-31', freq='D')
values = [10, 20, 15, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175, 180, 185, 190,
195, 200, 205, 210, 215, 220, 225, 230, 235, 240, 245, 250, 255, 260, 265, 270, 275, 280, 285, 290, 295, 300]

fig = go.Figure(data=go.Scatter(x=dates, y=values))
fig.update_layout(title='Customized Time Series Plot', xaxis_title='Date', yaxis_title='Values',
                  xaxis=dict(
                      rangeselector=dict(
                          buttons=list([
                              dict(count=1, label='1d', step='day', stepmode='backward'),
                              dict(count=7, label='1w', step='day', stepmode='backward'),
                              dict(count=1, label='1m', step='month', stepmode='backward'),
                              dict(count=6, label='6m', step='month', stepmode='backward'),
                              dict(count=1, label='YTD', step='year', stepmode='todate'),
                              dict(count=1, label='1y', step='year', stepmode='backward'),
                              dict(step='all')
                          ])
                      ),
                      rangeslider=dict(visible=True),
                      type='date'
                  ))
fig.show()
```

Output

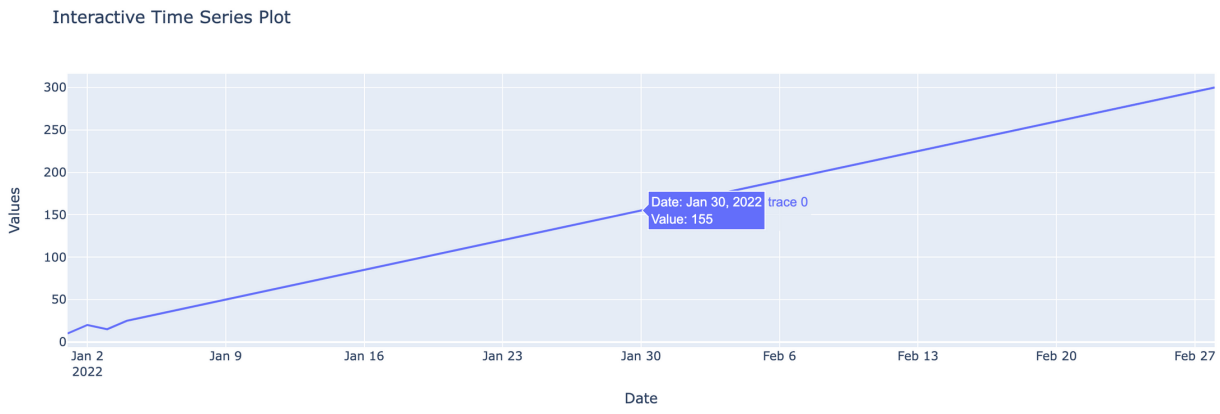


10.3 Interactive Features for Time Series

Plotly provides interactive features for time series plots, such as zooming, panning, and hover information. These features allow users to explore and analyze the data more effectively. The following example demonstrates adding interactive features to a time series plot:

```
import plotly.graph_objects as go
import pandas as pd

dates = pd.date_range(start='2022-01-01', end='2022-12-31', freq='D')
values = [10, 20, 15, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175, 180, 185, 190,
195, 200, 205, 210, 215, 220, 225, 230, 235, 240, 245, 250, 255, 260, 265, 270, 275, 280, 285, 290, 295, 300]

fig = go.Figure(data=go.Scatter(x=dates, y=values))
fig.update_layout(title='Interactive Time Series Plot', xaxis_title='Date', yaxis_title='Values')
fig.update_traces(hovertemplate='Date: %{x}<br>Value: %{y}')
fig.show()
```

Output



10.4 Animation with Plotly

Plotly allows users to create animations in time series plots, enabling the visualization of changes over time. Users can specify the duration, frame values, and frame labels. The following example demonstrates creating an animated time series plot:

```
import plotly.graph_objects as go
import pandas as pd

dates = pd.date_range(start='2022-01-01', end='2022-12-31', freq='D')
values = [10, 20, 15, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175, 180, 185, 190,
195, 200, 205, 210, 215, 220, 225, 230, 235, 240, 245, 250, 255, 260, 265, 270, 275, 280, 285, 290, 295, 300]

frames = []
for i in range(len(dates)):
    frames.append(go.Frame(data=[go.Scatter(x=dates[:i+1], y=values[:i+1])]))

fig = go.Figure(data=go.Scatter(x=dates, y=values, mode='markers'))
fig.frames = frames
fig.update_layout(title='Animated Time Series Plot', xaxis_title='Date', yaxis_title='Values')
fig.show()
```

Output

CHAPTER N.11

Advanced Features and Integration

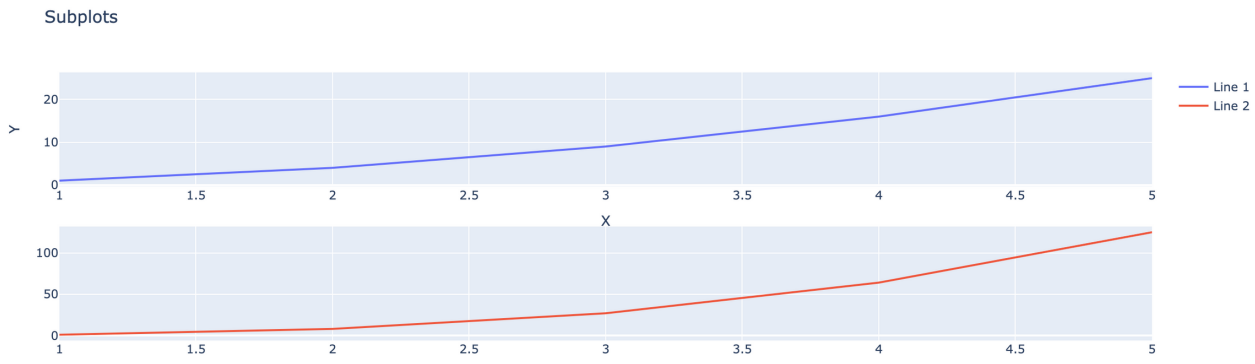
A Step-by-Step Guide

11.1 Subplots and Grids

Plotly allows users to create subplots and grids to display multiple plots together. This is useful for comparing different charts or visualizing multiple variables. The following example demonstrates creating subplots:

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

x = [1, 2, 3, 4, 5]
y1 = [1, 4, 9, 16, 25]
y2 = [1, 8, 27, 64, 125]

fig = make_subplots(rows=2, cols=1)

fig.add_trace(go.Scatter(x=x, y=y1, mode='lines', name='Line 1'), row=1, col=1)
fig.add_trace(go.Scatter(x=x, y=y2, mode='lines', name='Line 2'), row=2, col=1)

fig.update_layout(title='Subplots', xaxis_title='X', yaxis_title='Y')
fig.show()
```

Output



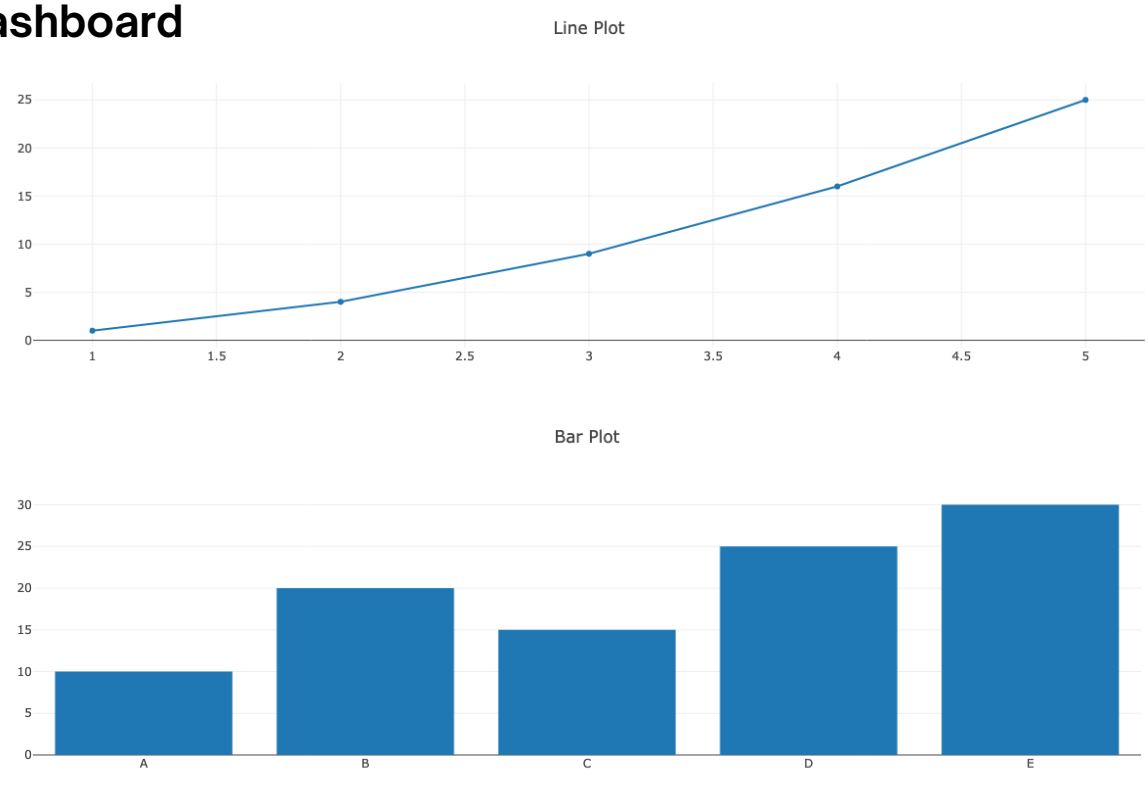
11.2 Dashboards with Plotly

Plotly can be used to create interactive dashboards and web applications through its Dash framework. Dash allows users to build interactive web applications using Python, HTML, and CSS. The following example demonstrates creating a simple dashboard with Plotly Dash:

```
import dash
import dash_core_components as dcc
import dash_html_components as html

app = dash.Dash(__name__)

app.layout = html.Div(children=[
    html.H1('Dashboard'),
    dcc.Graph(id='line-plot',
              figure={
                  'data': [{'x': [1, 2, 3, 4, 5], 'y': [1, 4, 9, 16, 25], 'type': 'line'}],
                  'layout': {'title': 'Line Plot'}
              }),
    dcc.Graph(id='bar-plot',
              figure={
                  'data': [{'x': ['A', 'B', 'C', 'D', 'E'], 'y': [10, 20, 15, 25, 30], 'type': 'bar'}],
                  'layout': {'title': 'Bar Plot'}
              })
])

if __name__ == '__main__':
    app.run_server(debug=True)
```

Output

Dashboard



Conclusion

Plotly is a powerful data visualization library that provides a wide range of charts and interactive features for data science. This practical guide covered the basics of Plotly, including line plots, scatter plots, bar plots, histograms, box plots, pie charts, heatmaps, time series plots, and advanced features such as subplots and integration with Dash. With this knowledge, you can leverage Plotly to create informative and visually appealing visualizations for your data science projects.