3. Exploratory Data Analysis

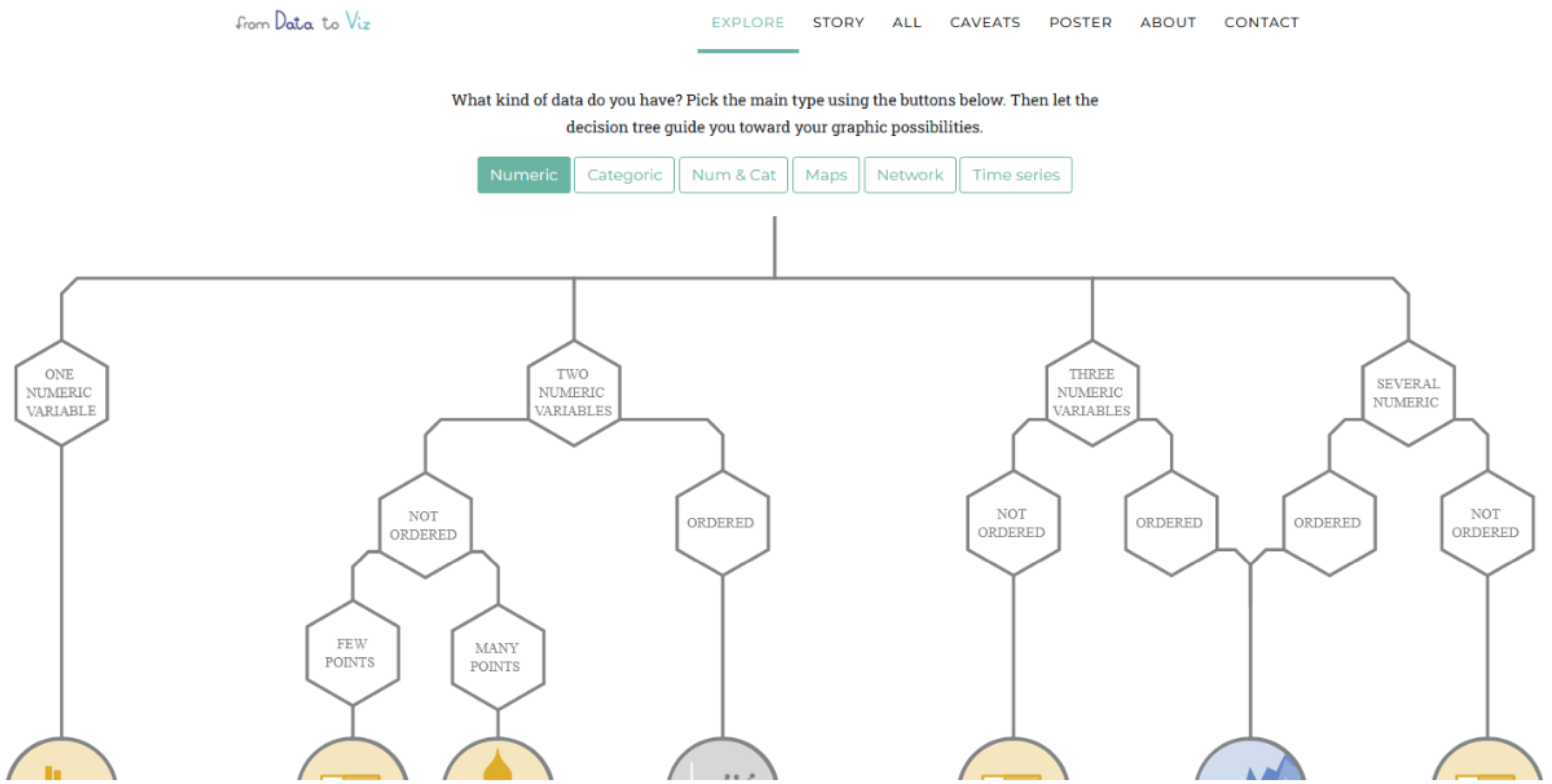
Exploratory Data Analysis

One of the fundamental parts of EDA is data visualization. This is the process of creating graphical representations of data in order to make it easier to understand and analyze, and so one can communicate their findings clearly and effectively to themselves and others. It can be used to identify patterns, trends, and relationships in data that might not be immediately obvious from looking at raw data. Python provides a wide range of libraries for data visualization, including Matplotlib, Seaborn, and Plotly.

We will be using Matplotlib and Plotly for plotting. We will be using all the data files up till now to plot various kinds of plots as an exercise to explain differnent concepts. These notes are only a guideline on how to do generic plots, you will be required to create plots from all the data we have used so far!

Visualization

It is important to know what kind of plots are used for different data and what insights you are trying to extract. This sight is a useful resource. This website data-to-viz.com provides a guide for data visualization. It helps data scientists to choose the right chart or plot to display their data, depending on the type and structure of the data they are working with. The website presents a wide range of data visualization options and provides examples and code snippets for creating them using different data visualization libraries such as ggplot2, Matplotlib, Seaborn etc in Python, R or JavaScript.



What kind of data do you have? Pick the main type using the buttons below. Then let the decision tree guide you toward your graphic possibilities.

Line Plot

Line plots are suitable for showing trends over time or a sequence. They help visualize patterns and changes in a variable over a continuous interval. They are useful for time series data or data with a natural ordering.

Matplotlib:

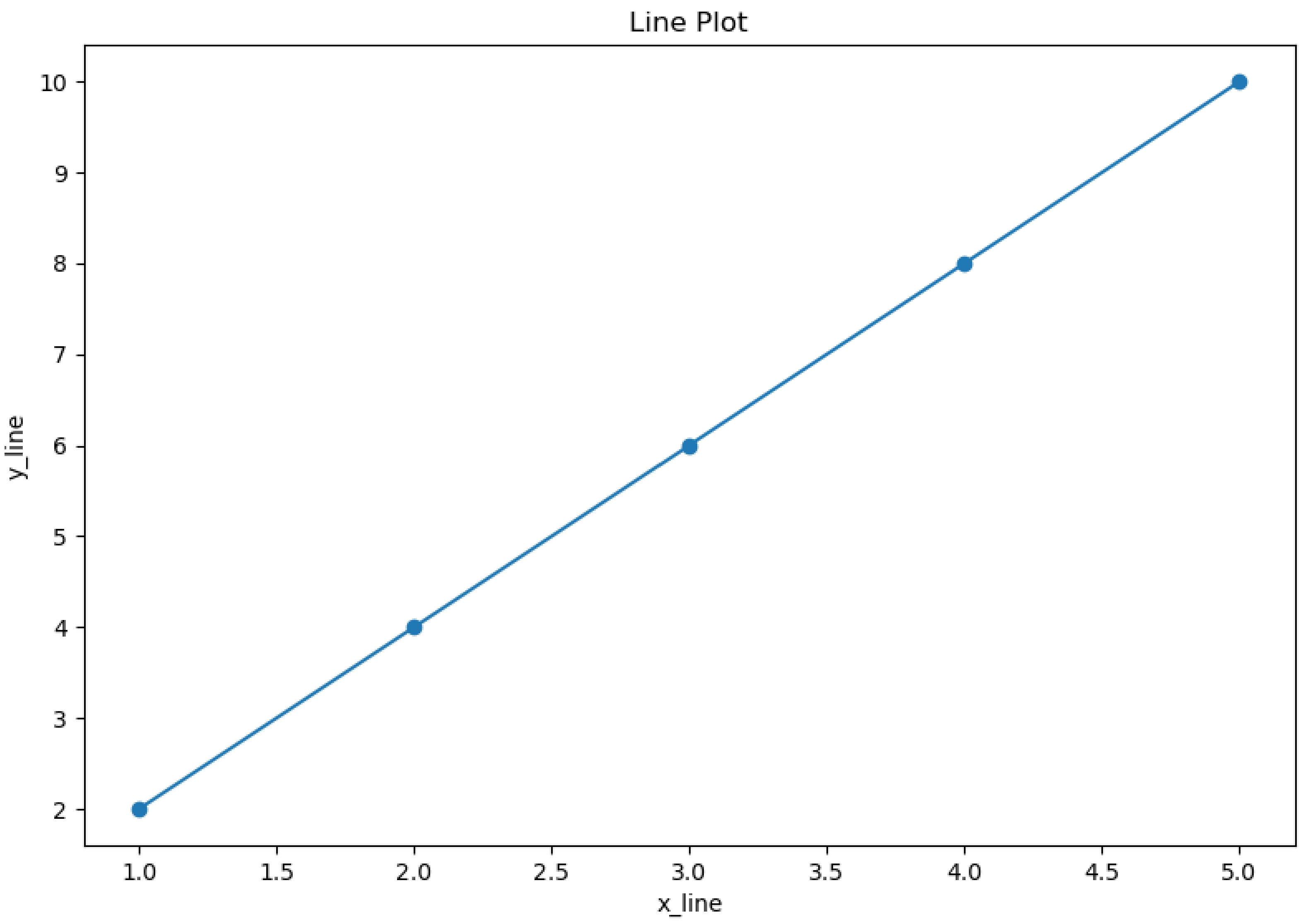
```
import matplotlib.pyplot as plt

x_line = [1, 2, 3, 4, 5];
y_line = [2, 4, 6, 8, 10];



plt.plot(x_line, y_line, '-o')
plt.xlabel("x_line")
plt.ylabel("y_line")

plt.title('Line Plot')
plt.show()
```



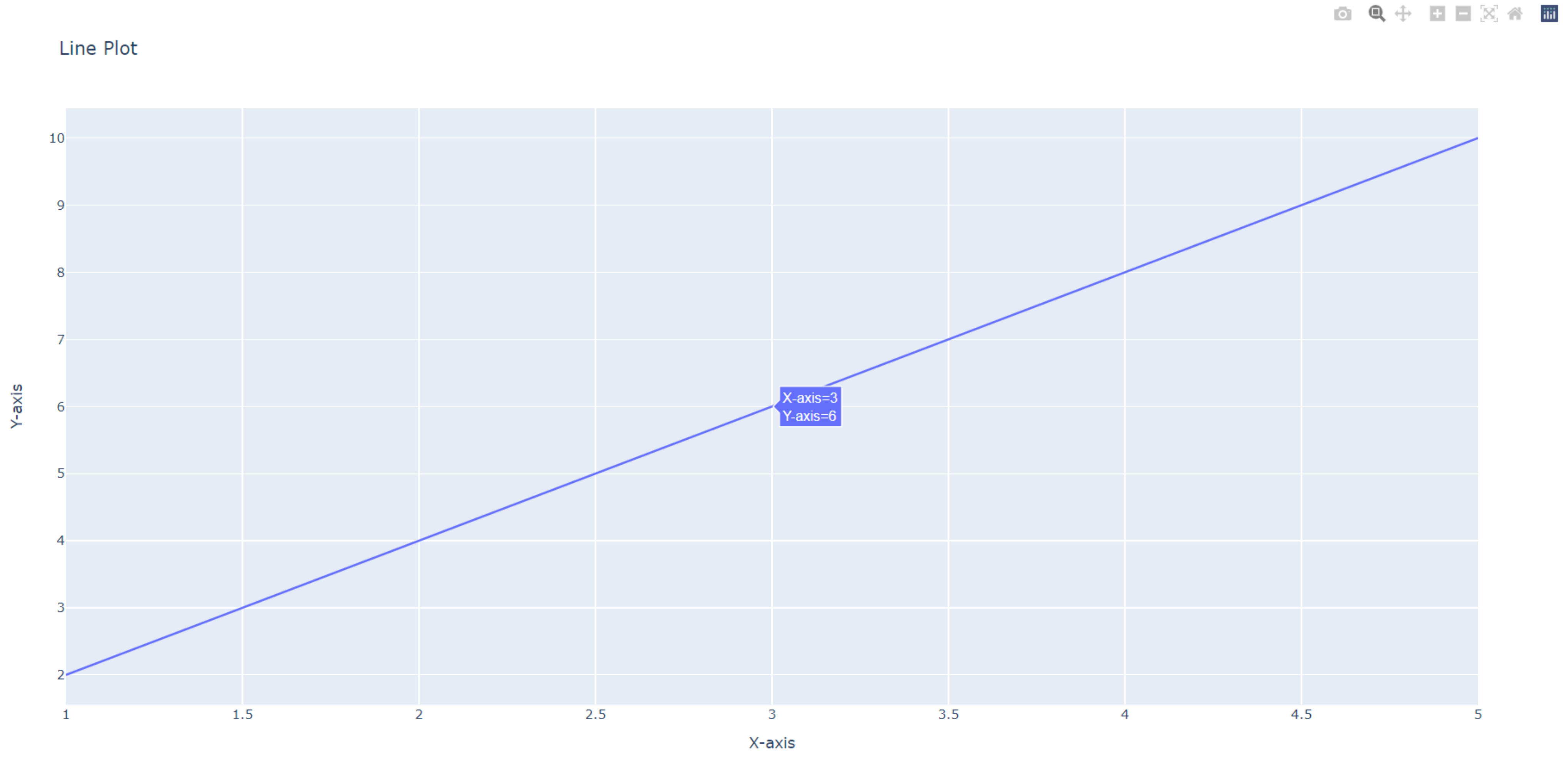
Plotly:

```
import plotly.express as px
```

```
x_line = [1, 2, 3, 4, 5]
y_line = [2, 4, 6, 8, 10]

fig = px.line(x=x_line, y=y_line, labels={'x': 'X-axis', 'y': 'Y-axis'}, title='Line Plot')
fig.write_html("plot.html")

# This is used to automatically open up a browser of your plot
import webbrowser
webbrowser.open("plot.html")
```



Bar Plot

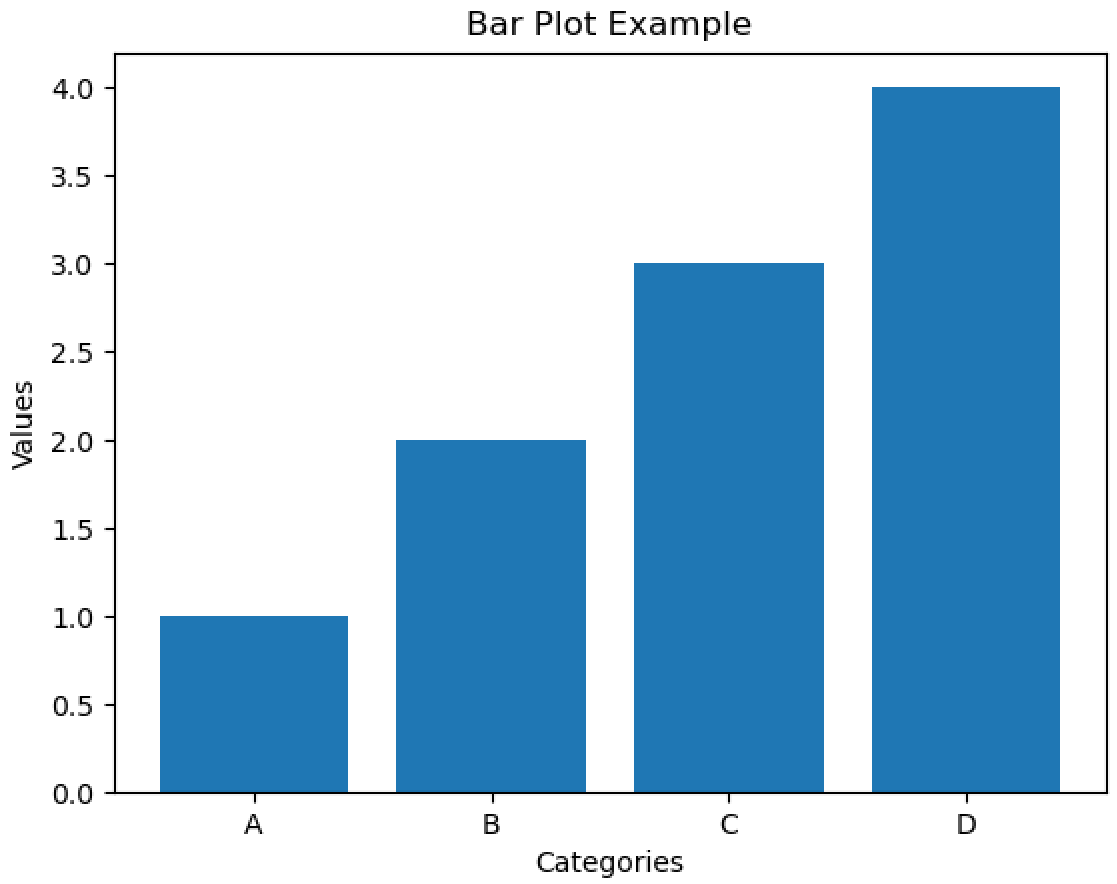
Bar charts are effective for visualizing the distribution of categorical variables. They help compare the frequencies or proportions of different categories. They are useful for exploring the distribution of a categorical variable, such as the count of observations in different groups.

Matplotlib:

```
import matplotlib.pyplot as plt
```

```
x_bar = ['A', 'B', 'C', 'D']
y_bar = [1, 2, 3, 4]

plt.bar(x_bar, y_bar)
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Plot Example')
plt.show()
```

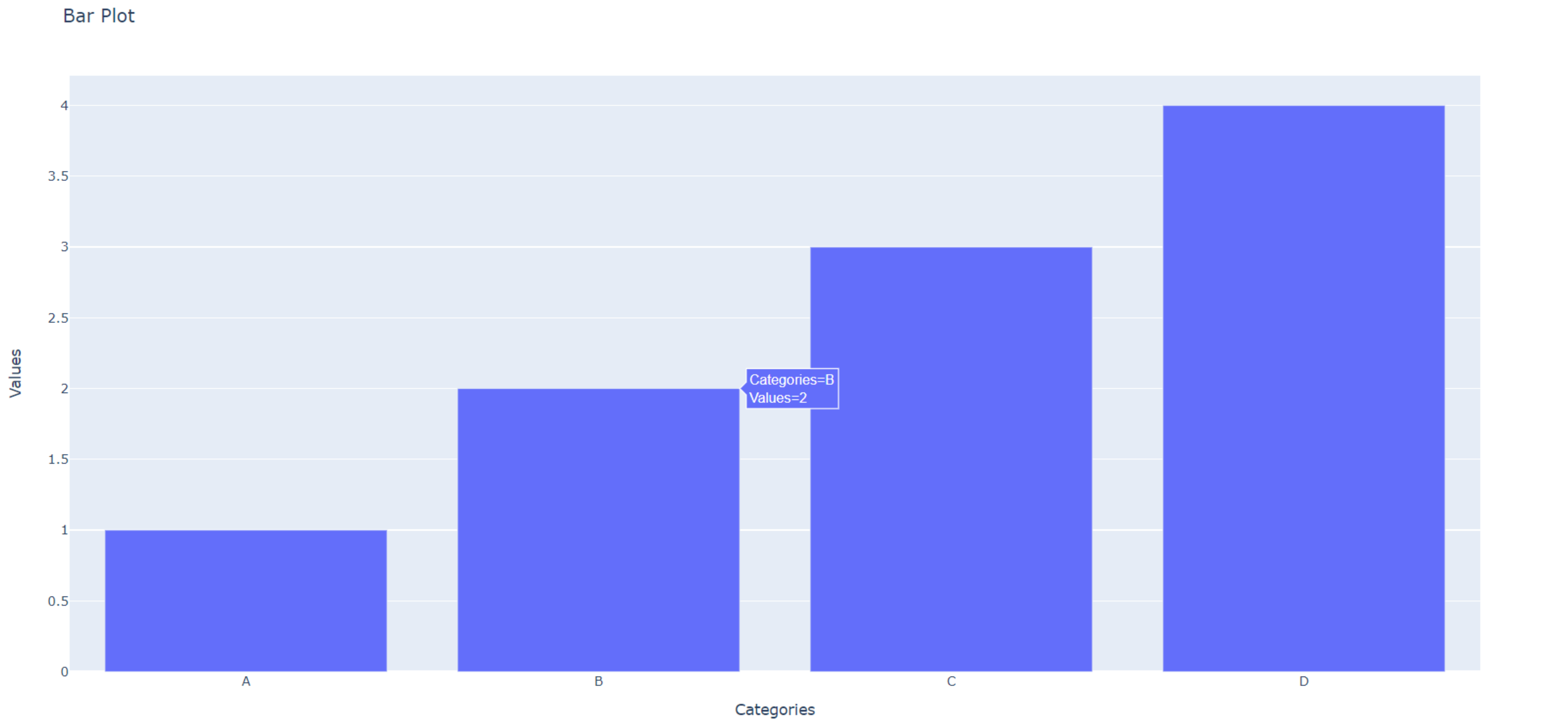


Plotly:

```
import plotly.express as px

x_bar = ['A', 'B', 'C', 'D']
y_bar = [1, 2, 3, 4]
fig = px.bar(x=x_bar, y=y_bar, labels={'x': 'Categories', 'y': 'Values'}, title='Bar Plot')
fig.write_html("plot.html")

# This is used to automatically open up a browser of your plot
import webbrowser
webbrowser.open("plot.html")
```



Scatter Plot

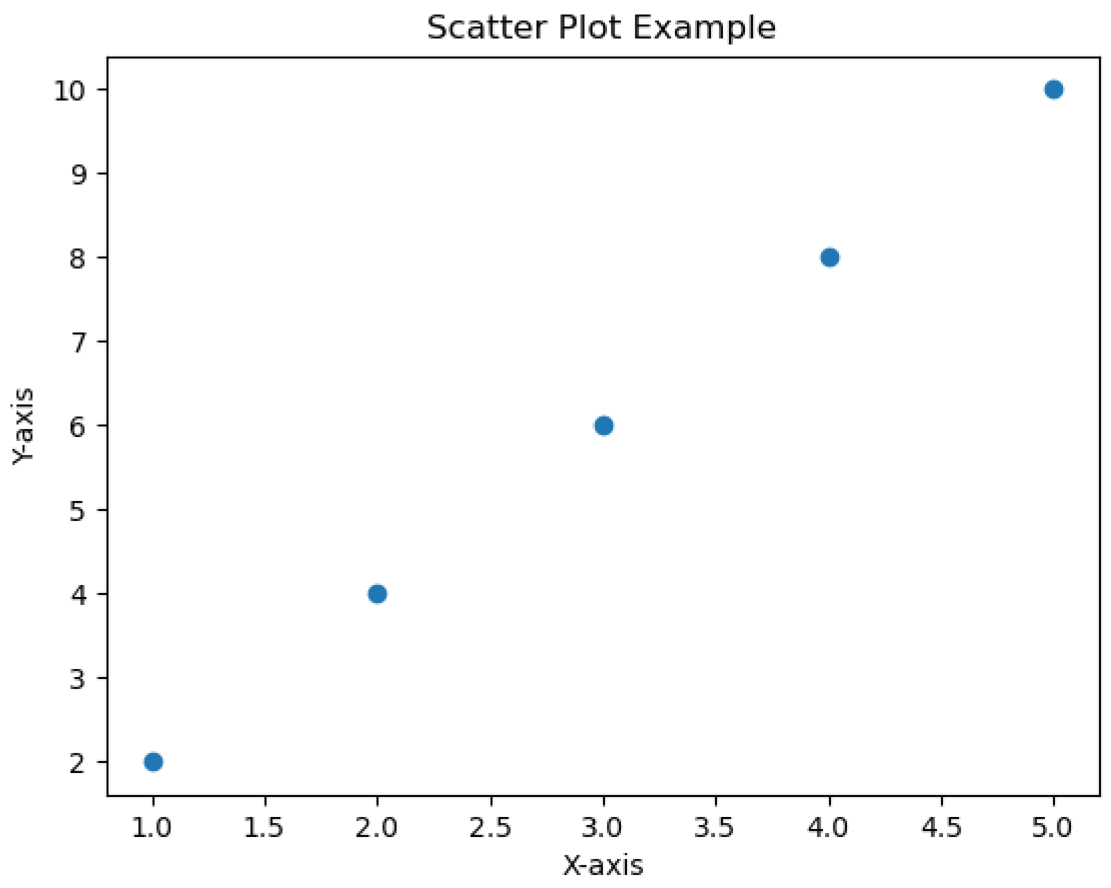
Scatter plots visualize the relationship between two numerical variables. They help identify patterns, correlations, and potential outliers. They are useful for exploring the correlation between variables and identifying trends or clusters.

Matplotlib:

```
import matplotlib.pyplot as plt
```

```
x_scatter = [1, 2, 3, 4, 5]
y_scatter = [2, 4, 6, 8, 10]

plt.scatter(x_scatter, y_scatter)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot Example')
plt.show()
```



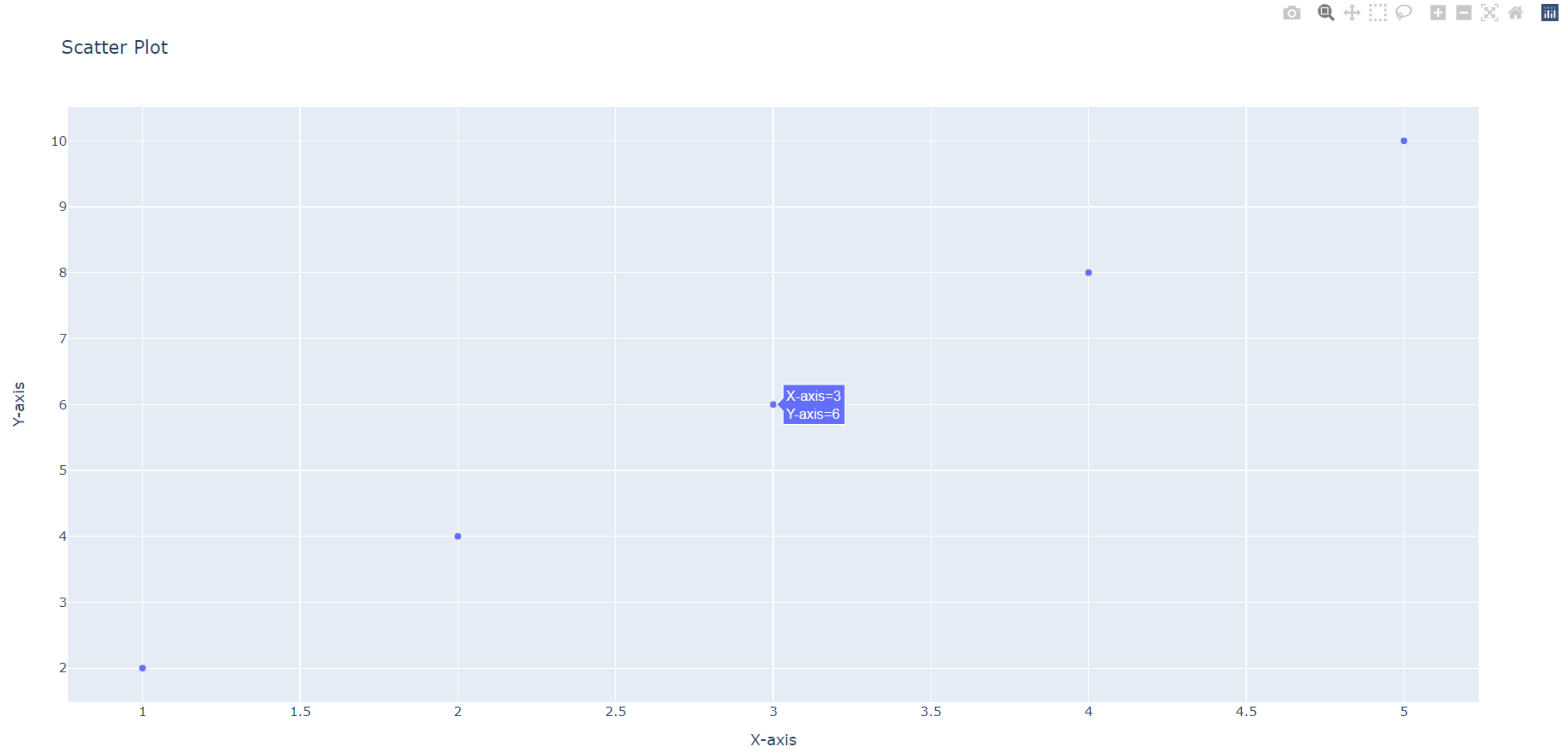
Plotly

```
import plotly.express as px

x_scatter = [1, 2, 3, 4, 5]
y_scatter = [2, 4, 6, 8, 10]

fig = px.scatter(x=x_scatter, y=y_scatter, labels={'x': 'X-axis', 'y': 'Y-axis'}, title='Sc
atter Plot')
fig.write_html("plot.html")

# This is used to automatically open up a browser of your plot
import webbrowser
webbrowser.open("plot.html")
```



Histogram Plot

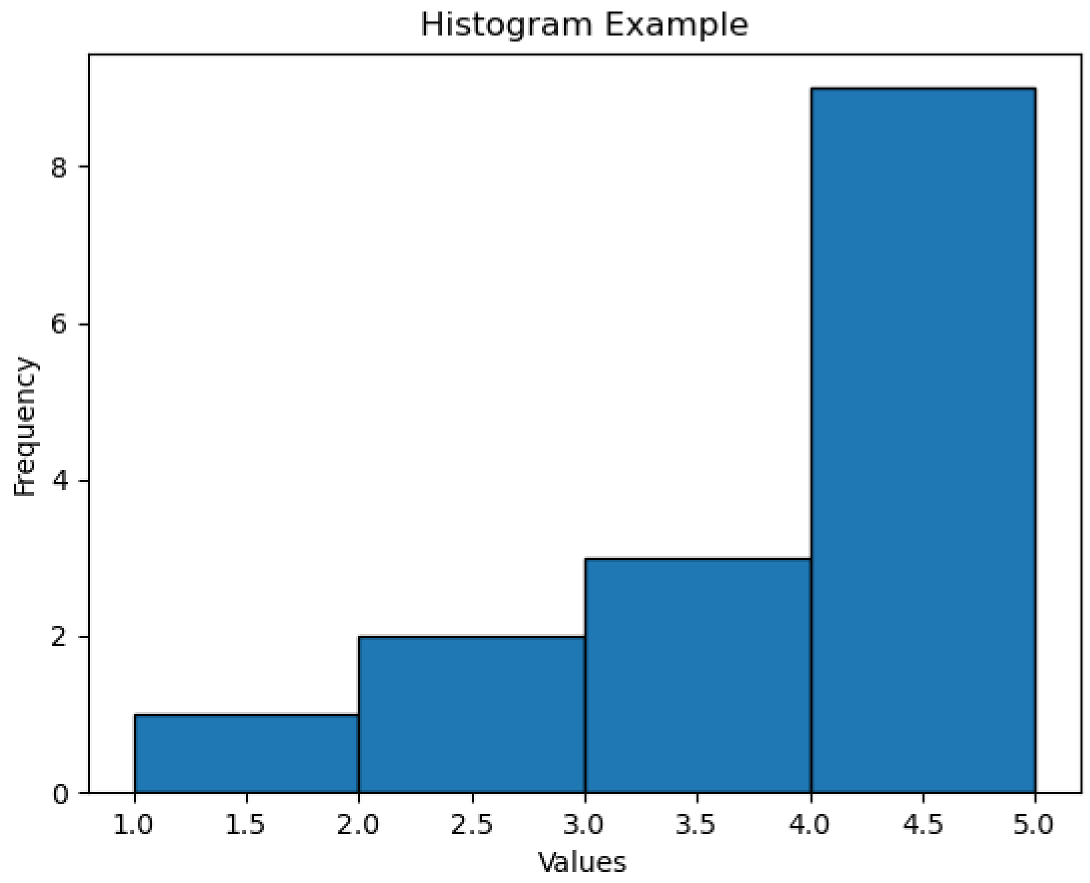
Histograms provide a visual representation of the distribution of a single variable. They help in understanding the central tendency, spread, and shape of the data. They are useful for exploring the distribution of numerical variables, such as age, income, or any continuous variable.

Matplotlib

```
import matplotlib.pyplot as plt
```

```
x_histogram = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5]

plt.hist(x_histogram, bins=range(min(x_histogram), max(x_histogram) + 1), edgecolor='blac
k')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Histogram Example')
plt.show()
```



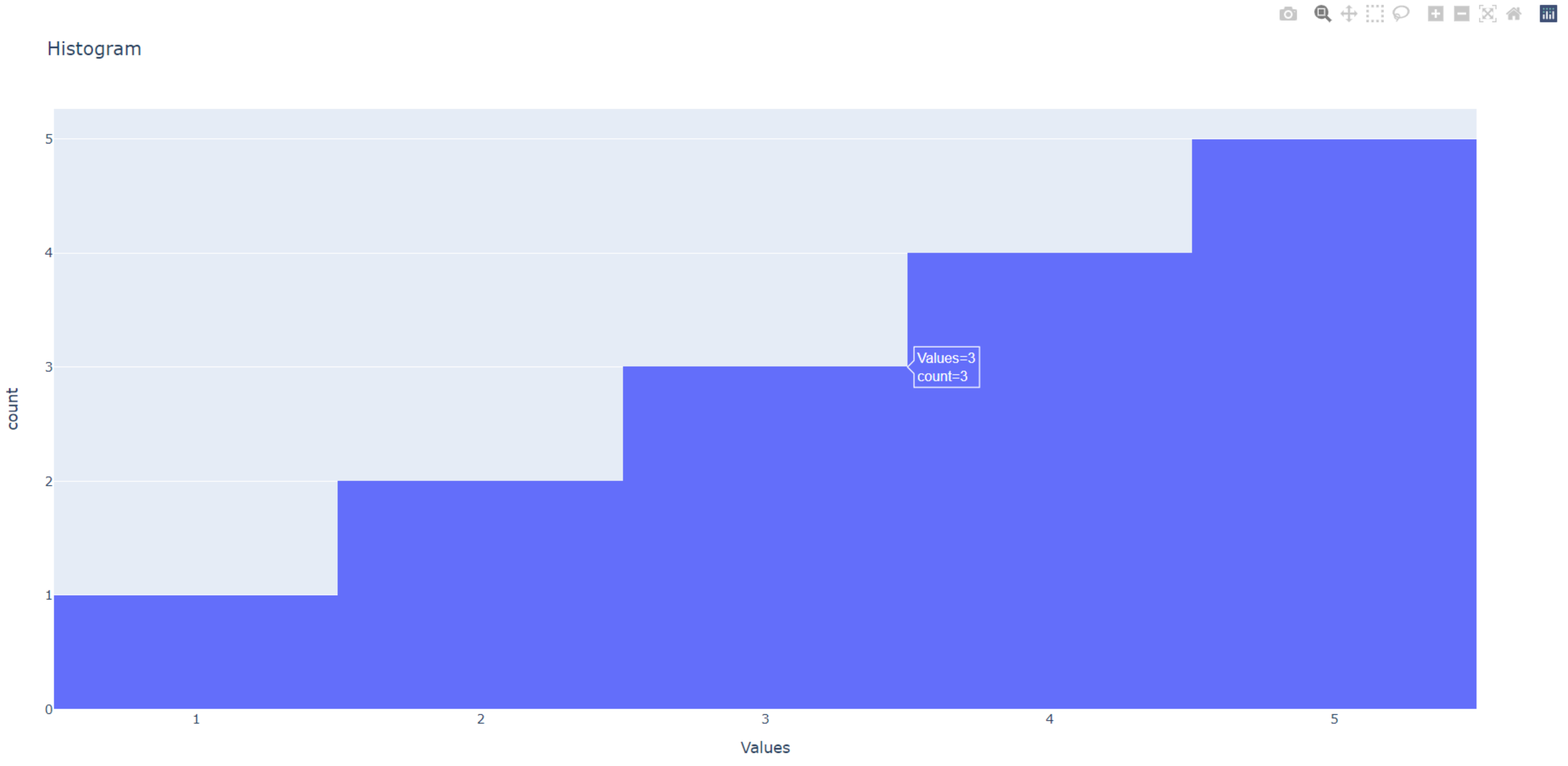
Plotly

```
import plotly.express as px

x_histogram = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5]

fig = px.histogram(x=x_histogram, labels={'x': 'Values'}, title='Histogram')
fig.write_html("plot.html")

# This is used to automatically open up a browser of your plot
import webbrowser
webbrowser.open("plot.html")
```



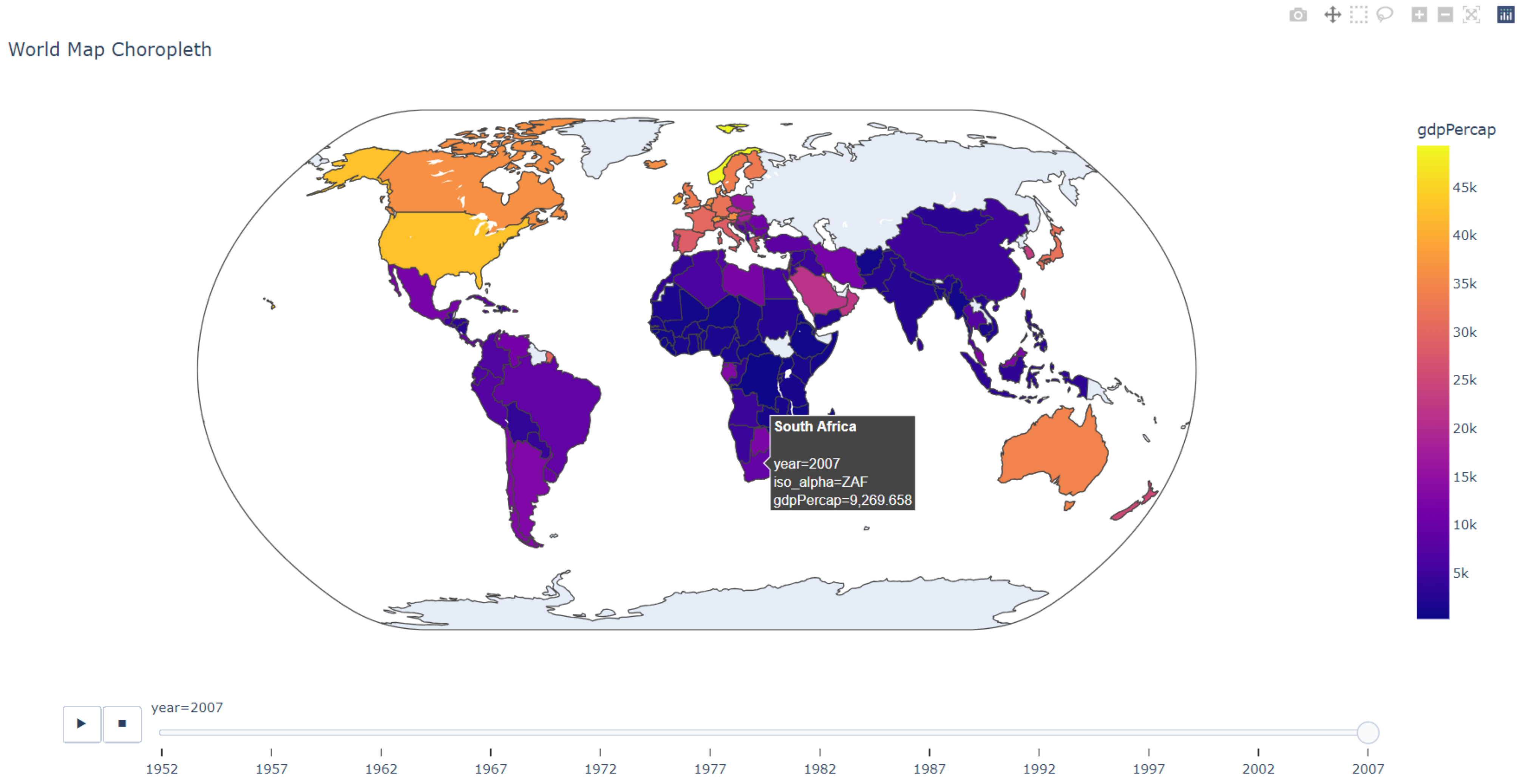
Maps

Map plots are visualizations that use geographical maps to represent data spatially. These plots are particularly useful when dealing with data that has a geographic component, such as regional sales, population distribution, or any information tied to specific locations.

```
data = px.data.gapminder()

# Create a choropleth world map
```

```
fig = px.choropleth(
    data_frame=data,
    locations="iso_alpha",
    color="gdpPercap",
    hover_name="country",
    animation_frame="year",
    title="World Map Choropleth",
    color_continuous_scale=px.colors.sequential.Plasma,
    projection="natural earth"
)
```



Combining Plots

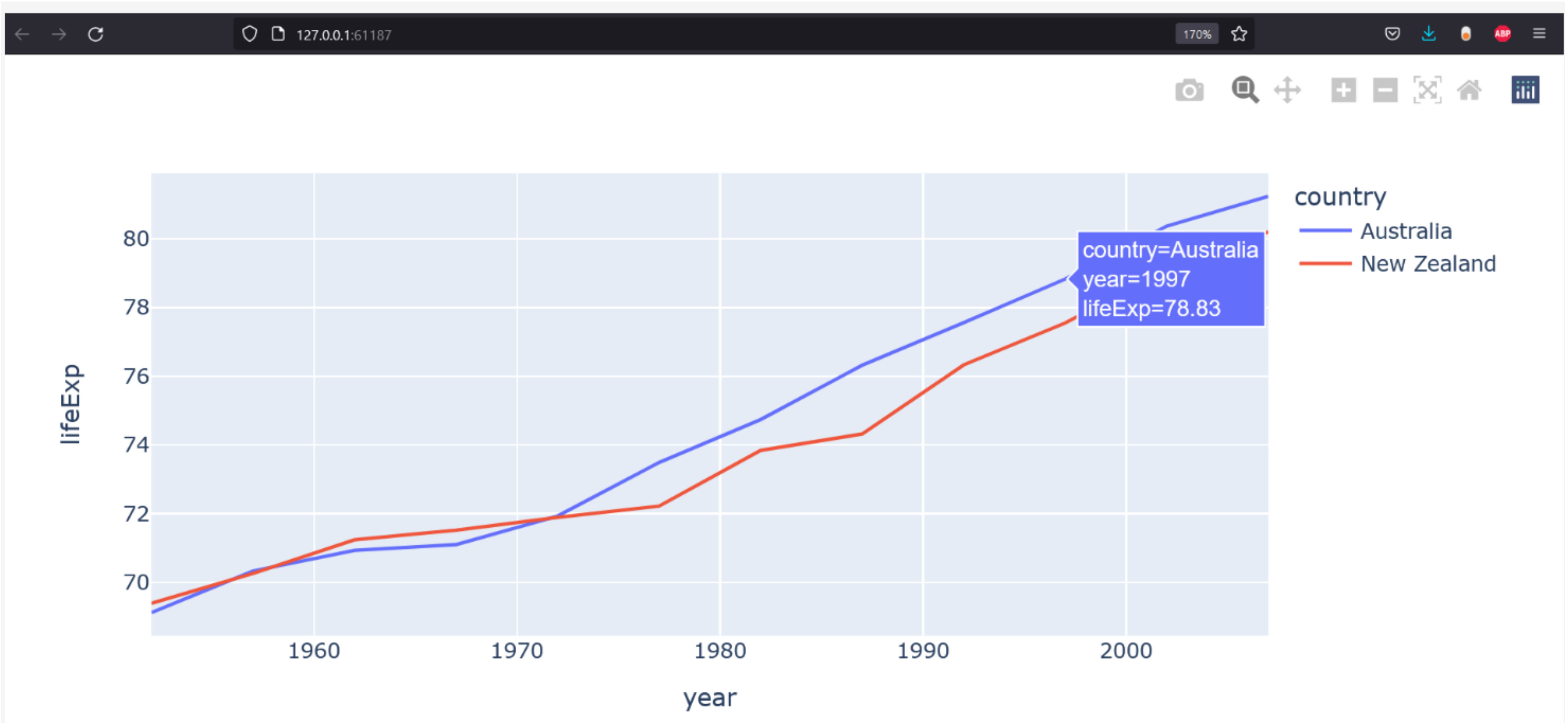
Example code:

```
import plotly.express as px

df = px.data.gapminder().query("continent=='Oceania'")
```

```
fig = px.line(df, x="year", y="lifeExp", color='country')
fig.write_html("plot.html")

# This is used to automatically open up a browser of your plot
import webbrowser
webbrowser.open("plot.html")
```



General EDA Guidelines:

1. Summary Statistics

Use Pandas features like .info and .describe to get identfiy key varibales and dataset statistics. Check data types, nulls, and count for various columns.

3. Data Visualization Techniques:

Familiarize yourself with histograms, box plots, scatter plots, and correlation matrices using seaborn and matplotlib. Note insights revealed by different visualizations.

3. Handling Missing Data and Outliers:

Learn techniques for missing data handling and outlier identification using pandas. Note the impact of missing data on analysis.

4. Univariate and Bivariate Analysis:

Explore individual variable characteristics and relationships using descriptive statistics and visualizations. Analyze bivariate relationships through scatter plots, pair plots, and correlation analysis.

5. Categorical Data Analysis:

Analyze and visualize categorical data using bar charts, pie charts, and count plots. Note the significance of categorical data.

6. Time Series Analysis:

Analyze time series data using line plots, seasonal decomposition, and autocorrelation plots. Note challenges in time series analysis.

Pandas Profiling

*** Something cool to look forward to ***