Data Visualization

Kittikun Jitpairod

Introduction to Data Visualization in Python

Data visualization is a crucial part of data analysis and communication. It allows us to represent data graphically, making it easier to identify patterns, trends, and outliers. Python offers several powerful libraries for data visualization, with Matplotlib and Plotly being two of the most popular. Let's explore the concept of data visualization and how these libraries can be used to create effective visualizations.

Why Data Visualization?

1. **Understand Data Quickly:** Visualizations can help you grasp large amounts of complex data at a glance.
2. **Identify Patterns and Trends:** Visual representations make it easier to spot trends, correlations, and patterns in data.
3. **Communicate Insights Effectively:** Visuals can convey information more quickly and memorably than text or raw numbers.
4. **Explore and Analyze Data:** Interactive visualizations allow for data exploration and can lead to new insights.

Key Principles of Data Visualization

1. **Clarity:** The visualization should clearly convey the intended information.
2. **Accuracy:** The visual representation must accurately reflect the underlying data.
3. **Efficiency:** Use the minimum amount of visual elements to convey the maximum amount of information.
4. **Aesthetics:** While not the primary goal, an aesthetically pleasing visualization can enhance engagement and comprehension.

Python Libraries for Data Visualization

Python offers several libraries for creating data visualizations. Two of the most popular are Matplotlib and Plotly:

Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It provides a MATLAB-like interface and is known for its flexibility and fine-grained control.

Key features:

- Suitable for publication-quality figures
- Highly customizable
- Good for static images and basic interactivity
- Integrates well with NumPy and pandas

Plotly

Plotly is a modern, interactive plotting library that creates web-based visualizations. It's excellent for creating interactive and web-ready charts and plots.

Key features:

- Creates interactive, web-based visualizations
- Excellent for dashboards and web applications
- Supports a wide range of chart types
- Offers both high-level (Plotly Express) and low-level APIs

Import Libraries

First, let's import the necessary libraries:

Matplotlib

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

Plotly

```
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd
import numpy as np

# import this when using notebook (Jupyter, Google Colab, ...)
import plotly.io as pio
pio.renderers.default = "notebook"
```

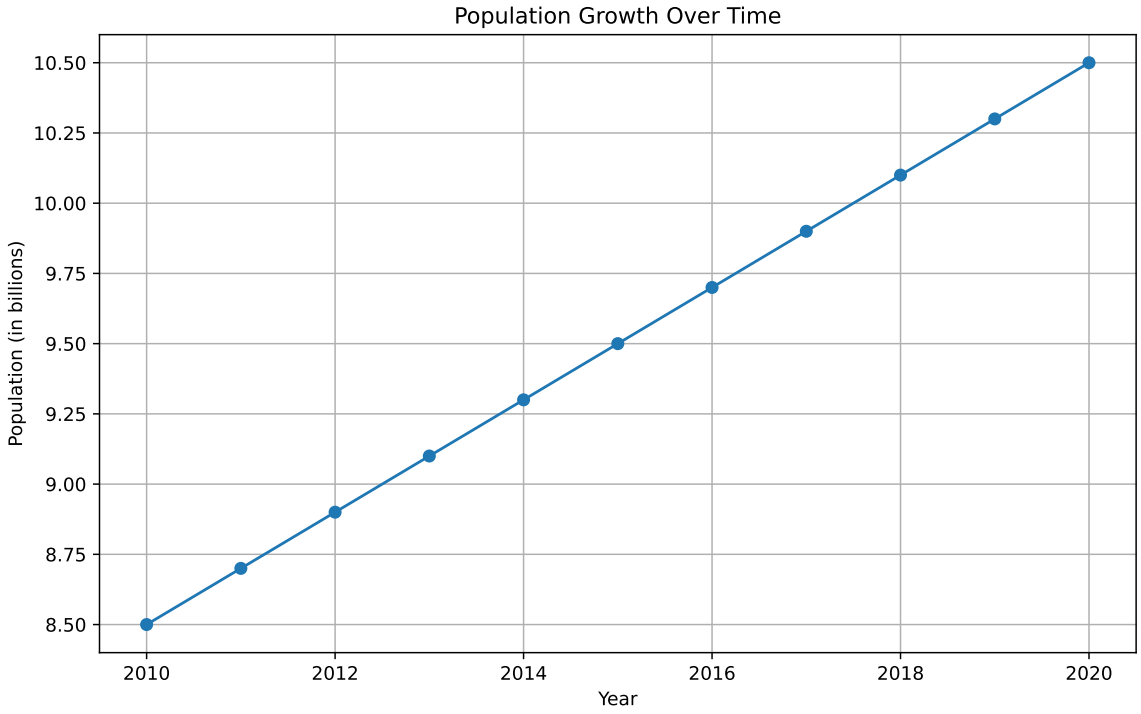
Line Plot

Line plots are great for showing trends over time or any continuous data.

```
years = np.arange(2010, 2021)
population = [8.5, 8.7, 8.9, 9.1,
              9.3, 9.5, 9.7, 9.9,
              10.1, 10.3, 10.5]

df = pd.DataFrame({'Year': years, 'Population': population})
```

Matplotlib

```
# Create the plot
plt.figure(figsize = (10, 6))
plt.plot(df['Year'], df['Population'], marker = 'o')

# Customize the layout
plt.title('Population Growth Over Time')
plt.xlabel('Year')
plt.ylabel('Population (in billions)')
plt.grid(True)

# Show the plot
plt.show()
```



Plotly

```
# Create the plot
fig = px.line(df, x = 'Year', y = 'Population', markers = True,
              title = 'Population Growth Over Time',
              labels = {'Population': 'Population (in billions)'},
              )

# Customize the layout
fig.update_layout(
    font = dict(size = 12),
    height = 600,
    width = 800
)

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

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

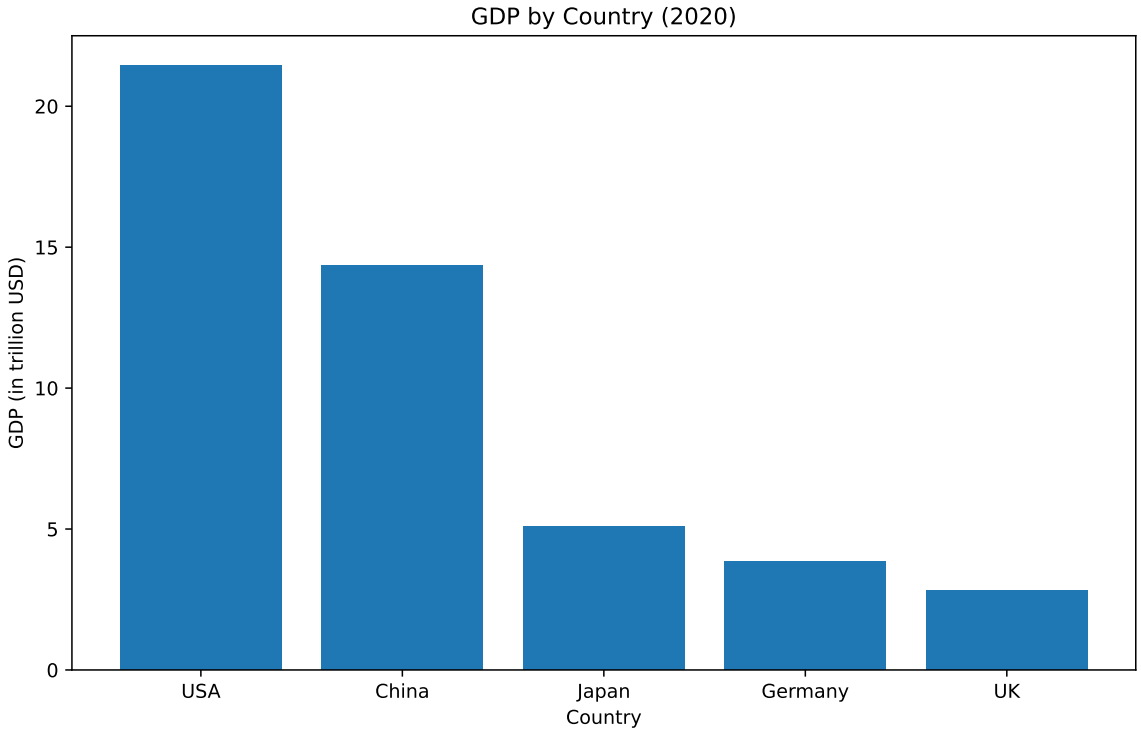
Vertical Bar Plot

Bar plots are useful for comparing quantities across different categories.

```
data = {
    'Country': ['USA', 'China', 'Japan', 'Germany', 'UK'],
    'GDP': [21.43, 14.34, 5.08, 3.86, 2.83]
}

df = pd.DataFrame(data)
```

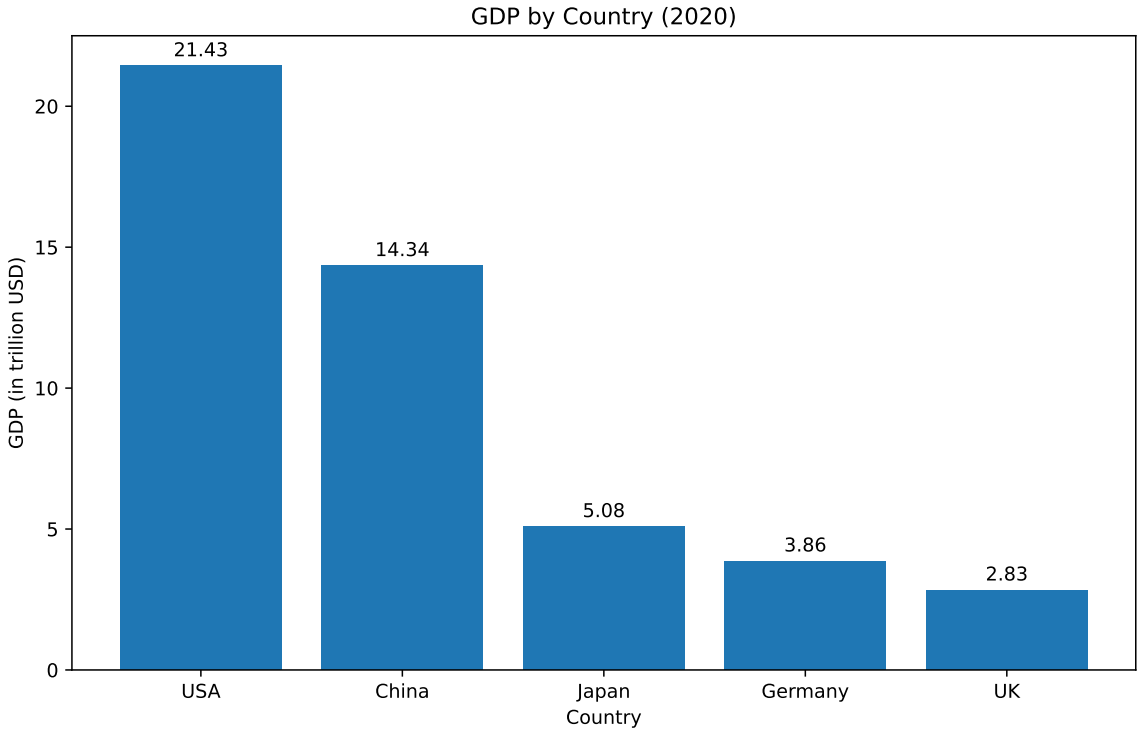
Matplotlib

```
# Create the plot
plt.figure(figsize = (10, 6))
plt.bar(df['Country'], df['GDP'])

# Customize the layout
plt.title('GDP by Country (2020)')
plt.xlabel('Country')
plt.ylabel('GDP (in trillion USD)')

# Show the plot
plt.show()
```



We can use the `ax.bar_label()` method, which is a more straightforward way, to add labels to bar plots.

```
fig, ax = plt.subplots(figsize = (10, 6))
bars = ax.bar(df['Country'], df['GDP'])
ax.set_title('GDP by Country (2020)')
ax.set_xlabel('Country')
ax.set_ylabel('GDP (in trillion USD)')

# Add value labels on the bars
ax.bar_label(bars, fmt = '%.2f', padding = 3)

plt.show()
```



Poltly

```
# Create the plot
fig = px.bar(df, x = 'Country', y = 'GDP',
             title = 'GDP by Country (2020)',
             labels = {'GDP': 'GDP (in trillion USD)'},
             text = 'GDP',  # This will display the GDP values on the bars
             height = 600, width = 800)

# Customize the layout
fig.update_traces(texttemplate = '%{text:.2f}', textposition = 'outside')
fig.update_layout(uniformtext_minsize = 8, uniformtext_mode = 'hide')

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

```
Unable to display output for mime type(s): text/html
```

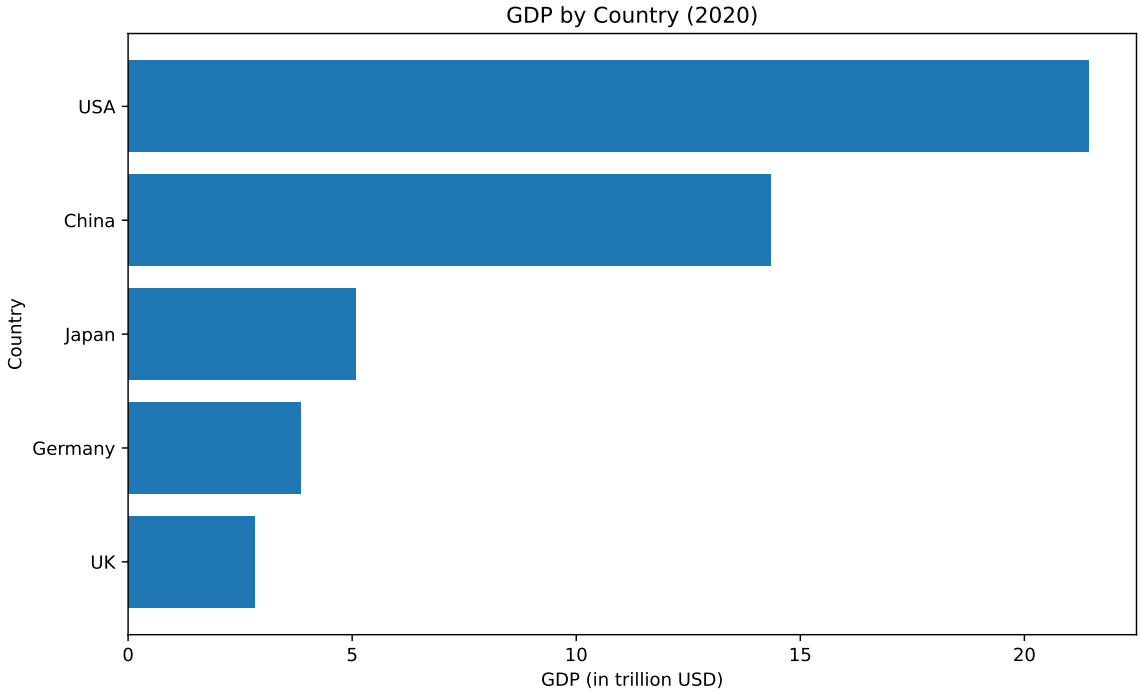
Horizontal Bar Plot

```
countries = ['China', 'Japan', 'USA',
             'Germany', 'UK']
gdp = [14.34, 5.08, 21.43, 3.86, 2.83]

df = pd.DataFrame({
    'Country': countries,
    'GDP': gdp,
})
df.sort_values(by = 'GDP', inplace = True)
print(df)
```

```
   Country    GDP
4       UK   2.83
3  Germany   3.86
1    Japan   5.08
0    China  14.34
2      USA  21.43
```

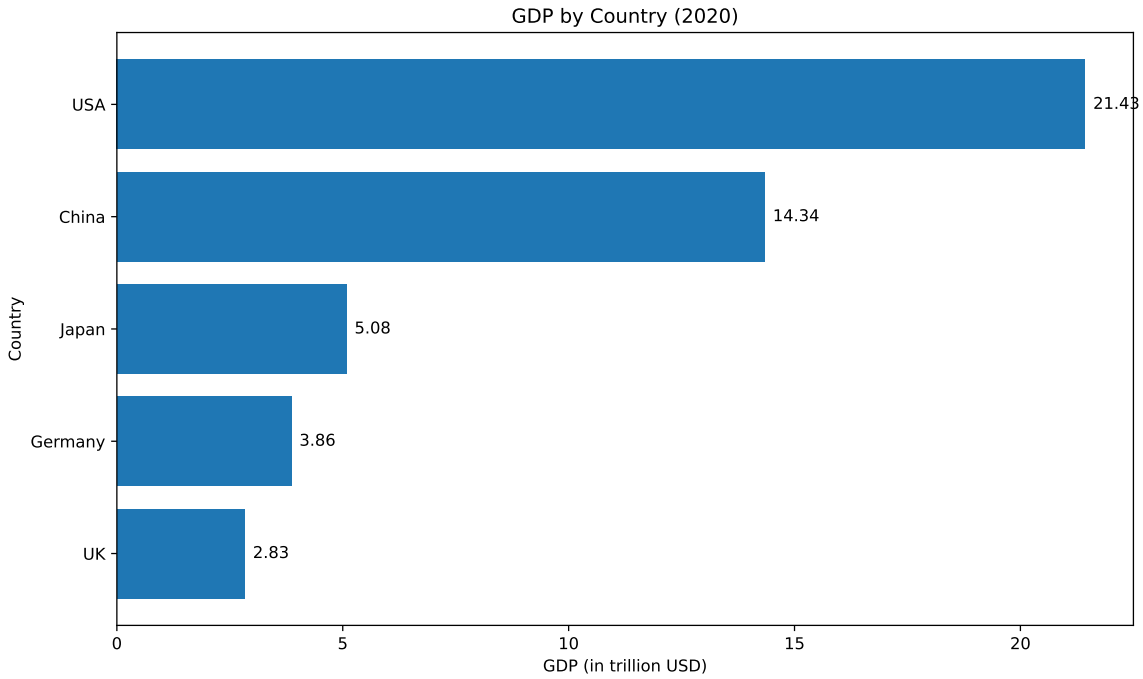
Matplotlib

```
# Create the plot
plt.figure(figsize = (10, 6))

# Customize the layout
plt.barh(df['Country'], df['GDP'])
plt.title('GDP by Country (2020)')
plt.xlabel('GDP (in trillion USD)')
plt.ylabel('Country')

# Show the plot
plt.show()
```



We can use the `ax.bar_label()` method, which is a more straightforward way, to add labels to bar plots.

```
# Create the plot
fig, ax = plt.subplots(figsize = (10, 6))
bars = ax.barh(df['Country'], df['GDP'])

# Add value labels on the bars
ax.bar_label(bars, fmt = '%.2f', padding = 5)

# Customize the plot
ax.set_title('GDP by Country (2020)')
ax.set_xlabel('GDP (in trillion USD)')
ax.set_ylabel('Country')

# Adjust layout to prevent clipping of labels
plt.tight_layout()

# Show the plot
plt.show()
```



Plotly

```
# Create the plot
fig = px.bar(df,
             x = 'GDP',
             y = 'Country',
             orientation = 'h',
             title = 'GDP by Country (2020)',
             labels = {'GDP': 'GDP (in trillion USD)'},
             text = 'GDP',
             height = 500,
             width = 800)

# Customize the layout
fig.update_traces(texttemplate = '%{text:.2f}', textposition = 'outside')
fig.update_layout(yaxis = {'categoryorder':'total ascending'})

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

```
Unable to display output for mime type(s): text/html
```

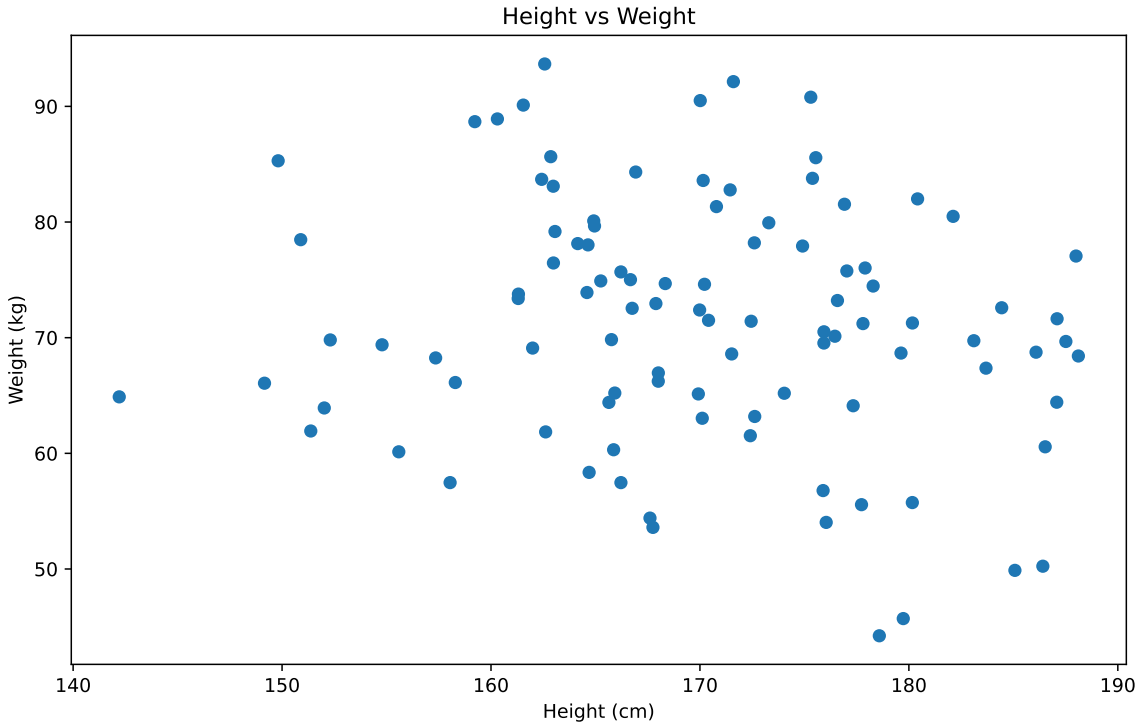
Scatter Plot

Scatter plots are excellent for showing the relationship between two variables.

```
height = np.random.normal(170, 10, 100)
weight = np.random.normal(70, 10, 100)

df = pd.DataFrame({
    'Height (cm)': height,
    'Weight (kg)': weight
})
```

Matplotlib

```
# Create the plot
plt.figure(figsize = (10, 6))
plt.scatter(height, weight)

# Customize the layout
plt.title('Height vs Weight')
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')

# Show the plot
plt.show()
```



Plotly

```
# Create the plot
fig = px.scatter(df, x = 'Height (cm)', y = 'Weight (kg)',
                 title = 'Height vs Weight',
                 labels = {'Height (cm)': 'Height (cm)', 'Weight (kg)': 'Weight (kg)'},
                 height = 600, width = 800)

# Customize the layout
fig.update_traces(marker = dict(size = 8))
fig.update_layout(
    title_font_size = 20,
    xaxis_title_font_size = 14,
    yaxis_title_font_size = 14
)

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

```
Unable to display output for mime type(s): text/html
```

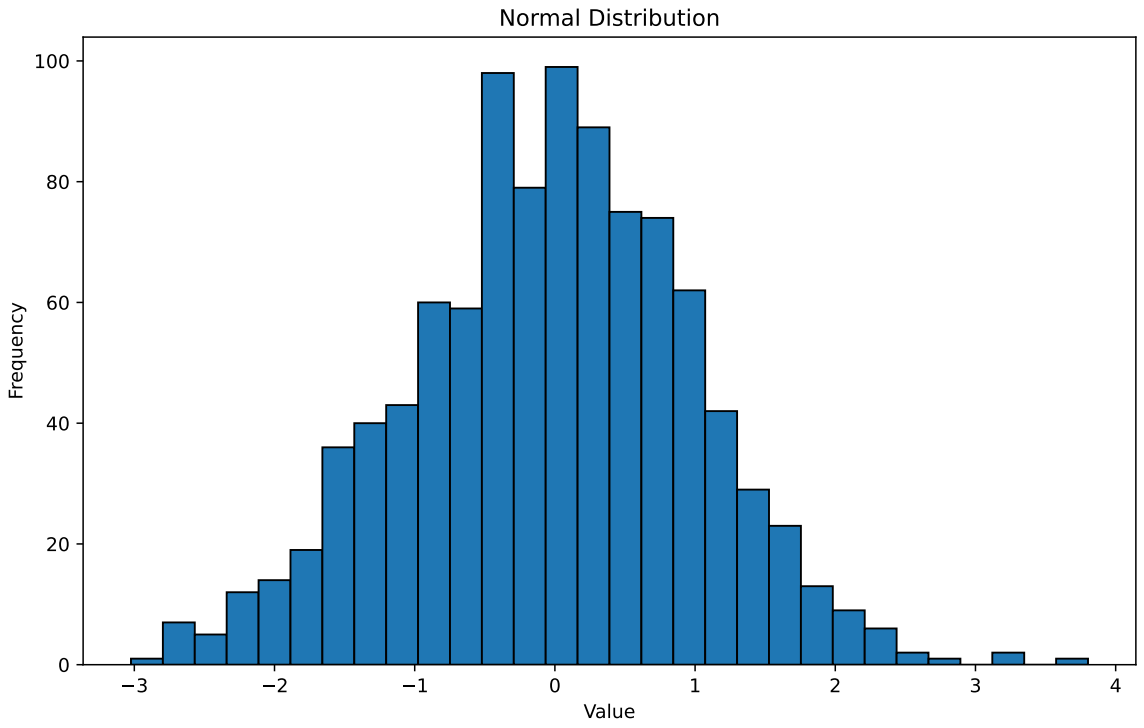
Histogram

Histograms show the distribution of a dataset.

```
data = np.random.normal(0, 1, 1000)
df = pd.DataFrame({'Value': data})
```

Matplotlib

```
# Create the plot
plt.figure(figsize = (10, 6))
plt.hist(data, bins = 30, edgecolor = 'black')

# Customize the layout
plt.title('Normal Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Show the plot
plt.show()
```



Plotly

```
# Create the plot
fig = px.histogram(df, x = 'Value',
                   nbins = 30,
                   title = 'Normal Distribution',
                   labels = {'Value': 'Value', 'count': 'Frequency'},
                   height = 600, width = 800)

# Customize the layout
fig.update_traces(marker_line_color = "black", marker_line_width = 1)
fig.update_layout(
    title_font_size = 20,
    xaxis_title_font_size = 14,
    yaxis_title_font_size = 14
)

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

```
Unable to display output for mime type(s): text/html
```

Pie Chart

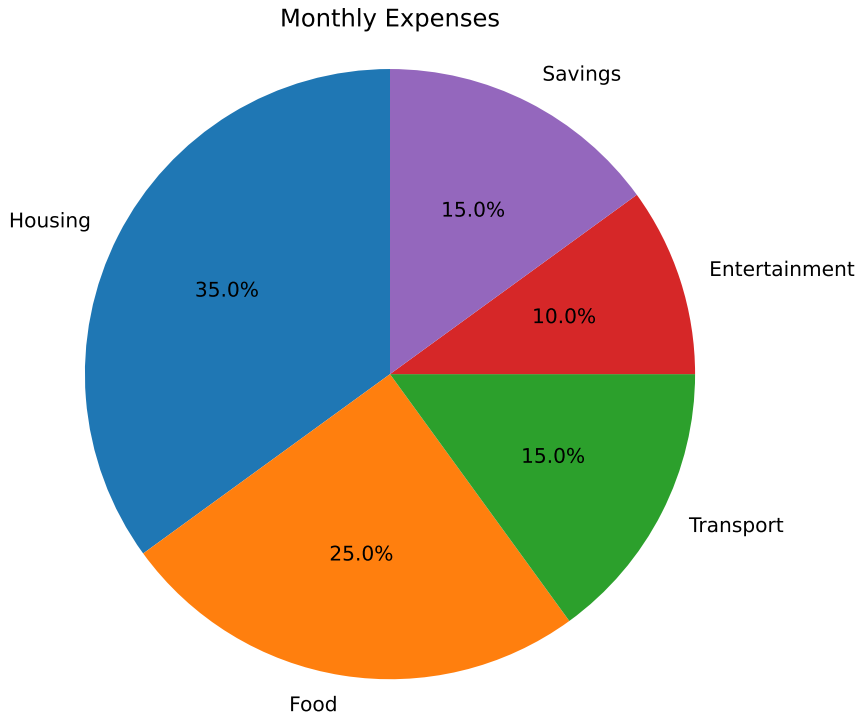
Pie charts are used to show proportions of a whole.

```
categories = ['Housing', 'Food',
    'Transport', 'Entertainment',
    'Savings']
expenses = [35, 25, 15, 10, 15]

df = pd.DataFrame({
    'Category': categories,
    'Expense': expenses
})
```

Matplotlib

```
# Create the plot
plt.figure(figsize = (10, 6))
plt.pie(expenses, labels = categories, autopct = '%1.1f%%', startangle = 90)
```

```
# Customize the layout
plt.title('Monthly Expenses')
plt.axis('equal')  # Equal aspect ratio

# Show the plot
plt.show()
```



Plotly

```
# Create the plot
fig = px.pie(df,
             values = 'Expense',
             names = 'Category',
             title = 'Monthly Expenses',
             height = 600,
             width = 800)

# Customize the layout
fig.update_traces(textposition = 'inside', textinfo = 'percent+label')
fig.update_layout(
    title_font_size = 20,
    legend_title_font_size = 14,
```

```
    uniformtext_minsize = 12,
    uniformtext_mode = 'hide'
)

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

```
Unable to display output for mime type(s): text/html
```

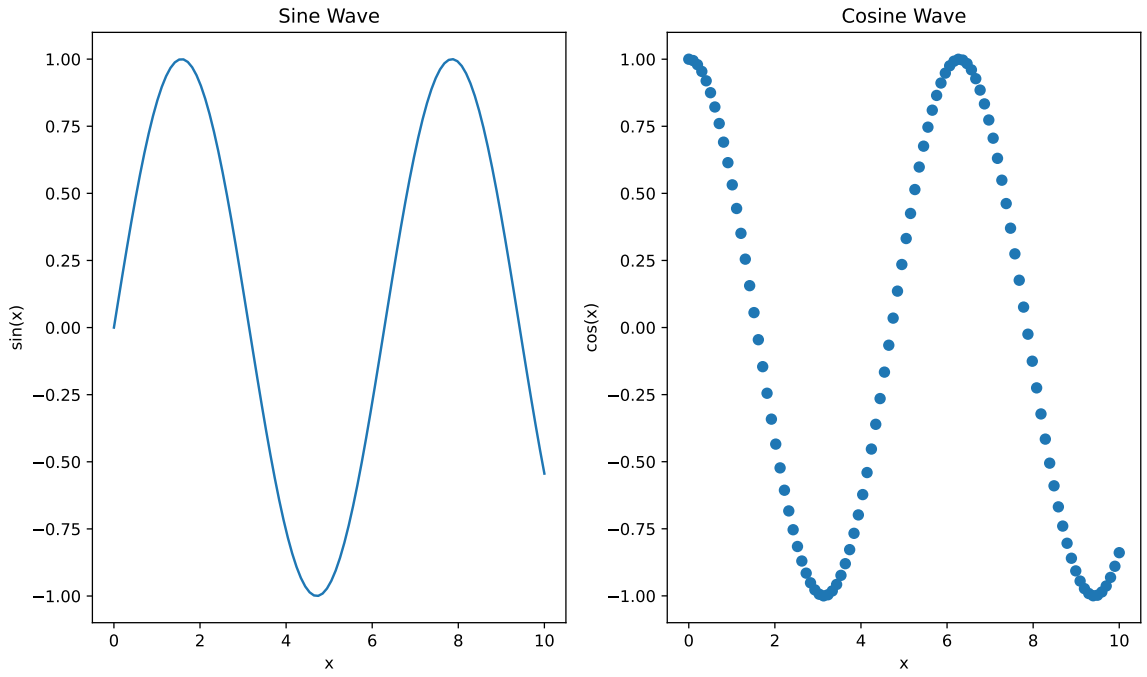
Subplots

Subplots allow you to combine multiple plots in one figure.

```
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
```

Matplotlib

```
fig, axs = plt.subplots(1, 2, figsize = (10,6))

axs[0].plot(x, y1)
axs[0].set_title('Sine Wave')
axs[0].set_xlabel('x')
axs[0].set_ylabel('sin(x)')

axs[1].scatter(x, y2)
axs[1].set_title('Cosine Wave')
axs[1].set_xlabel('x')
axs[1].set_ylabel('cos(x)')

plt.tight_layout()
plt.show()
```



Plotly

```
from plotly.subplots import make_subplots

# Create subplots
fig = make_subplots(rows = 1, cols = 2, subplot_titles = ("Sine Wave", "Cosine Wave"))

# Add traces for sine wave
fig.add_trace(
    go.Scatter(x = x, y = y1, name = "sin(x)"),
    row = 1, col = 1
)

# Add traces for cosine wave
fig.add_trace(
    go.Scatter(x = x, y = y2, name = "cos(x)", mode = "markers"),
    row = 1, col = 2
)

# Update layout
fig.update_layout(
    height = 600, width = 1000,
    title_text = "Sine and Cosine Waves",
    showlegend = False
```

```
)

# Update x and y axis labels
fig.update_xaxes(title_text = "x", row = 1, col = 1)
fig.update_xaxes(title_text = "x", row = 1, col = 2)
fig.update_yaxes(title_text = "sin(x)", row = 1, col = 1)
fig.update_yaxes(title_text = "cos(x)", row = 1, col = 2)

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

```
Unable to display output for mime type(s): text/html
```

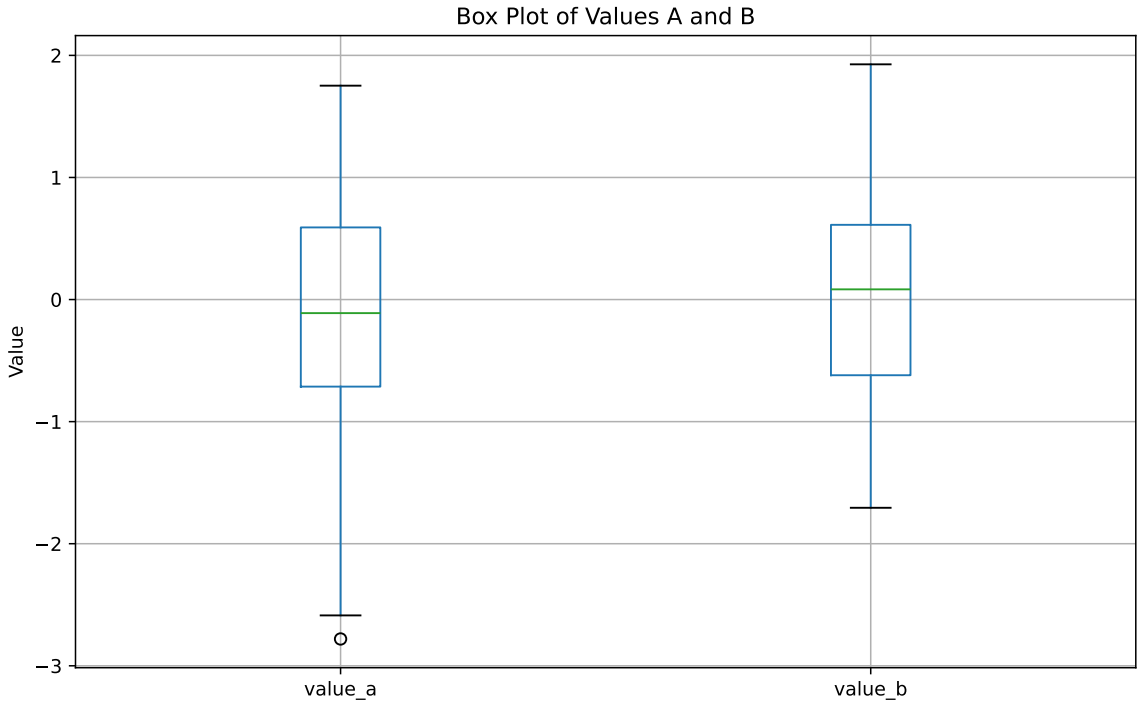
Box Plot

Box plots are useful for showing the distribution of data and identifying outliers:

```
df = pd.DataFrame({
    'value_a': np.random.randn(100),
    'value_b': np.random.randn(100),
})
```

Matplotlib

```
df.boxplot(column = ['value_a', 'value_b'], figsize = (10, 6))

plt.title('Box Plot of Values A and B')
plt.ylabel('Value')
plt.grid(True)
plt.show()
```



Plotly

```
# Melt the DataFrame to long format
df_melted = df.melt(var_name = 'Variable', value_name = 'Value')

# Create the box plot
fig = px.box(df_melted, x = 'Variable', y = 'Value',
             title = 'Box Plot of Values A and B',
             labels = {'Variable': 'Variable', 'Value': 'Value'},
             height = 600, width = 800)

# Customize the layout
fig.update_layout(
    title_font_size = 20,
    xaxis_title_font_size = 14,
    yaxis_title_font_size = 14,
    showlegend=False
)

# Add grid lines
fig.update_yaxes(showgrid = True, gridwidth = 1, gridcolor = 'LightGrey')
```

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

```
Unable to display output for mime type(s): text/html
```

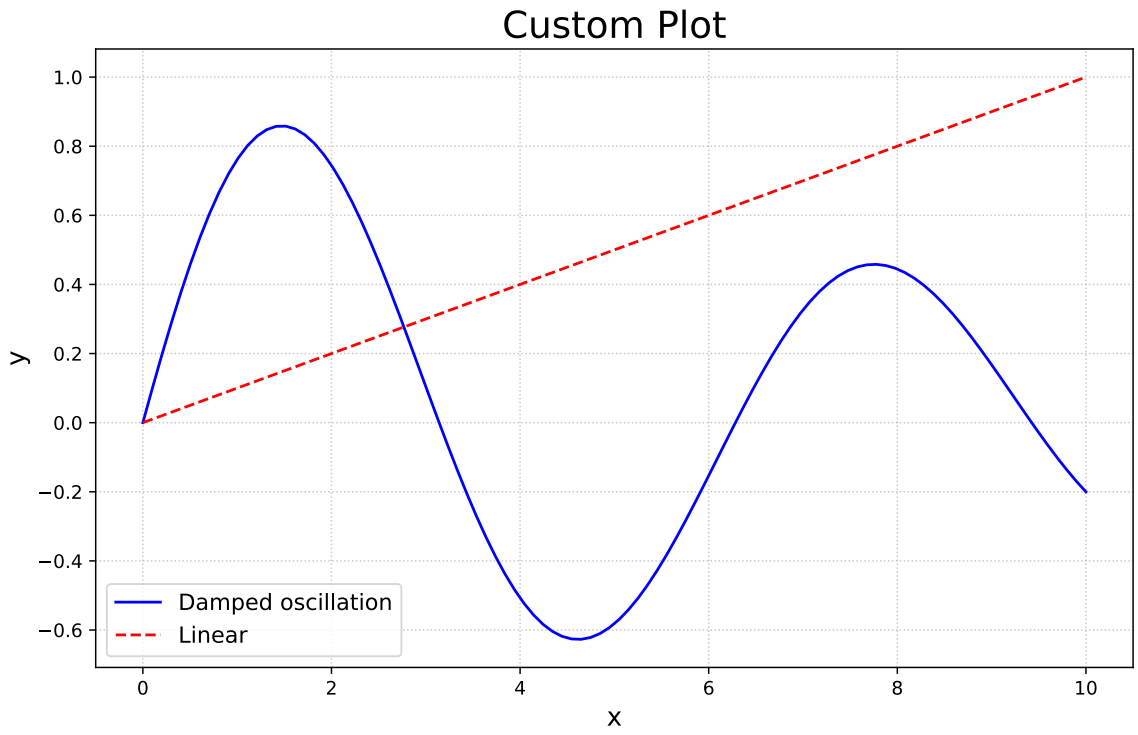
Customizing Plots

Matplotliband Plotly offers many customization options. Here's an example with a customized line plot:

```
x = np.linspace(0, 10, 100)
y1 = np.exp(-x/10)*np.sin(x)
y2 = x/10
```

Matplotlib

```
# Create the figure
plt.figure(figsize = (10, 6))

# Add traces
plt.plot(x, y1, 'b-', label = 'Damped oscillation')
plt.plot(x, y2, 'r--', label = 'Linear')

# Update layout
plt.title('Custom Plot', fontsize = 20)
plt.xlabel('x', fontsize = 14)
plt.ylabel('y', fontsize = 14)
plt.legend(fontsize = 12)

# Add grid
plt.grid(True, linestyle = ':', alpha = 0.7)

# Show the plot
plt.show()
```



Poltly

```
# Create the figure
fig = go.Figure()

# Add traces
fig.add_trace(go.Scatter(x = x, y = y1, mode = 'lines', name = 'Damped oscillation',
                         line = dict(color = 'blue', width = 2)))
fig.add_trace(go.Scatter(x = x, y = y2, mode = 'lines', name = 'Linear',
                         line = dict(color = 'red', width = 2, dash = 'dash')))

# Update layout
fig.update_layout(
    title = dict(text = 'Custom Plot', font = dict(size = 20)),
    xaxis = dict(title = 'x', titlefont = dict(size = 14)),
    yaxis = dict(title = 'y', titlefont = dict(size = 14)),
    legend = dict(font = dict(size = 12)),
    height = 600,
    width = 1000,
    hovermode = 'x unified',
)
```

```
# Add grid
fig.update_xaxes(showgrid = True, gridwidth = 1, gridcolor = 'rgba(0,0,0,0.1)')
fig.update_yaxes(showgrid = True, gridwidth = 1, gridcolor = 'rgba(0,0,0,0.1)')

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

```
Unable to display output for mime type(s): text/html
```

Exporting Plots

Export plot as image files.

Matplotlib

Saving as an image file: You can save your plot as an image file (PNG, JPG, SVG, etc.) using the `savefig()` method.

```
# Save as PNG
plt.savefig('matplotlib_1.png', dpi = 300, bbox_inches = 'tight')

# Save as SVG
plt.savefig('matplotlib_2.svg', bbox_inches = 'tight')
```

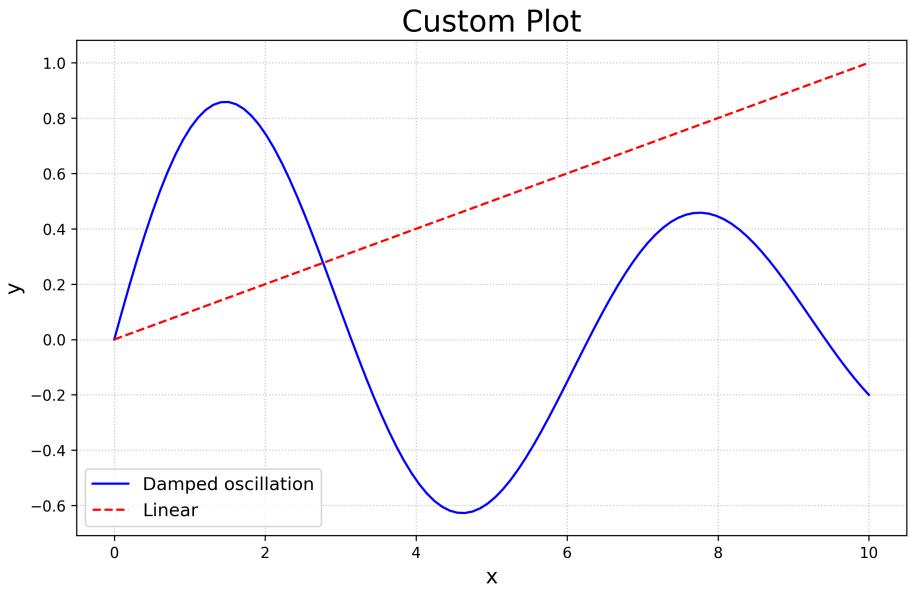


Figure 1: PNG

Plotly

Saving as a static image: You can save a Plotly figure as a static image using the `write_image()` method. This requires the `kaleido` package.

```
pip install kaleido
```

```
# Save as PNG
fig.write_image("plotly_1.png", scale = 2)

# Save as SVG
fig.write_image("plotly_2.svg")
```

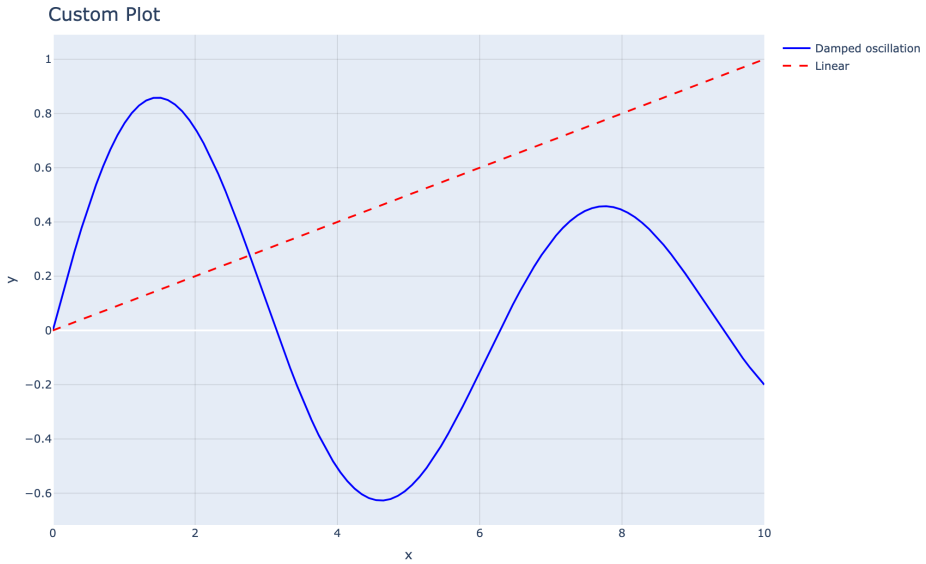


Figure 2: PNG