Lecture IX - Data Visualization

Programming with Python

Dr. Tobias Vlćek
Kühne Logistics University Hamburg - Fall 2025

Quick Recap of the last Lecture

Pandas: Data Analysis

- Powerful library for data manipulation and analysis
- Built on top of NumPy, providing additional functionality
- Key features of Pandas include:
  - Data loading from various file formats
  - Data cleaning and preprocessing
  - Powerful grouping and aggregation operations
  - Merging and joining datasets

Why NumPy and Pandas are Essential

- Basic tools for scientific computing and data analysis
- Efficient data structures and operations for large data
- Integration with other scientific Python libraries
- Used in data science, machine learning, and research

. . .

> Tip
>
> You might also need them in future lectures here!

Data Visualization

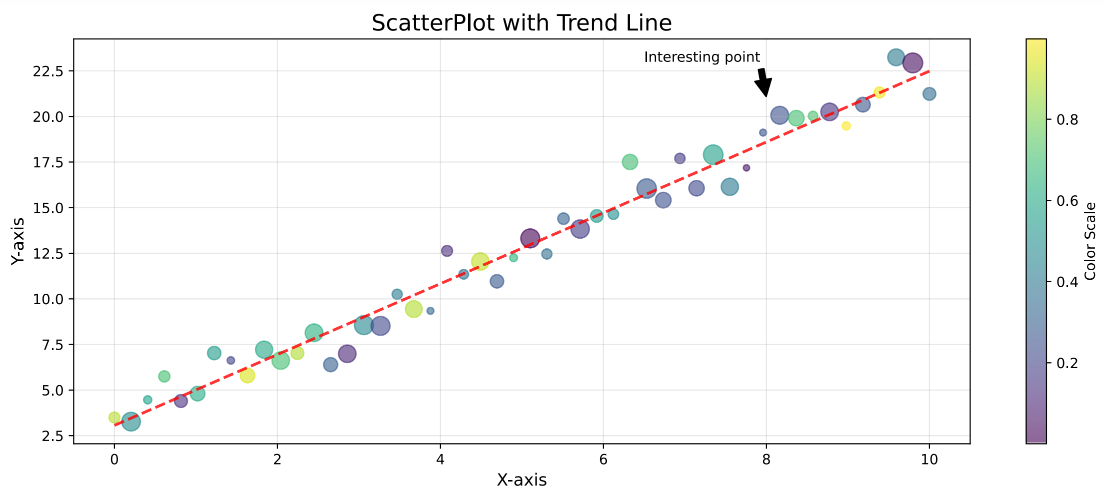
Question: What is

data visualization?

Visual Representations of Data

```python
import matplotlib.pyplot as plt
import numpy as np


# Generate data
np.random.seed(42)
```

```
x = np.linspace(0, 10, 50)
y = 3 + 2*x + np.random.randn(50)
sizes = np.random.randint(20, 200, 50)
colors = np.random.rand(50)

# Create the plot
plt.figure(figsize=(12, 5))
scatter = plt.scatter(x, y, c=colors, s=sizes, alpha=0.6, cmap='viridis')

# Add trend line
z = np.polyfit(x, y, 1)
p = np.poly1d(z)
plt.plot(x, p(x), "r--", alpha=0.8, linewidth=2)

# Customize the plot
plt.title("ScatterPlot with Trend Line", fontsize=16)
plt.xlabel("X-axis", fontsize=12)
plt.ylabel("Y-axis", fontsize=12)
plt.colorbar(scatter, label="Color Scale")

# Add a text annotation
plt.annotate("Interesting point", xy=(8, 21), xytext=(6.5, 23),
             arrowprops=dict(facecolor='black', shrink=0.05))

plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```



Importance of Data Visualization

- Communicates complex information clearly
- Helps in decision-making processes
- Reveals hidden patterns and relationships in data
- Makes data more accessible and engaging

...

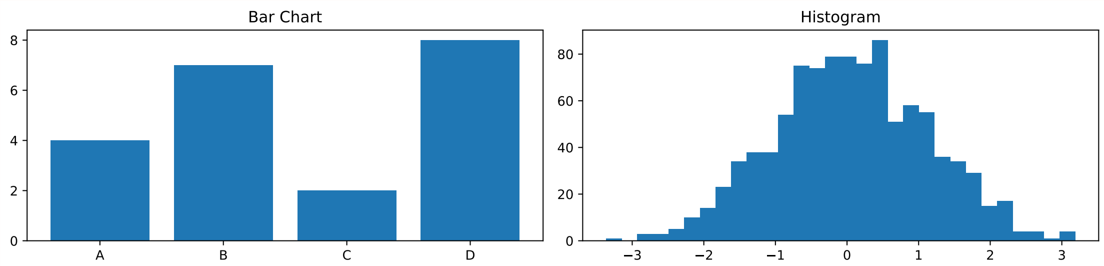
> Tip
>
> Helps to convice stakeholders!

Common Types of Data Visualizations

Bar Charts and Histograms

- Bar charts: Compare quantities across categories
- Histograms: Show distribution of a continuous variable

```python
import matplotlib.pyplot as plt
import numpy as np

# Bar chart
categories = ['A', 'B', 'C', 'D']
values = [4, 7, 2, 8]

plt.figure(figsize=(12, 3))
plt.subplot(121)
plt.bar(categories, values)
plt.title('Bar Chart')

# Histogram
data = np.random.randn(1000)

plt.subplot(122)
plt.hist(data, bins=30)
plt.title('Histogram')

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

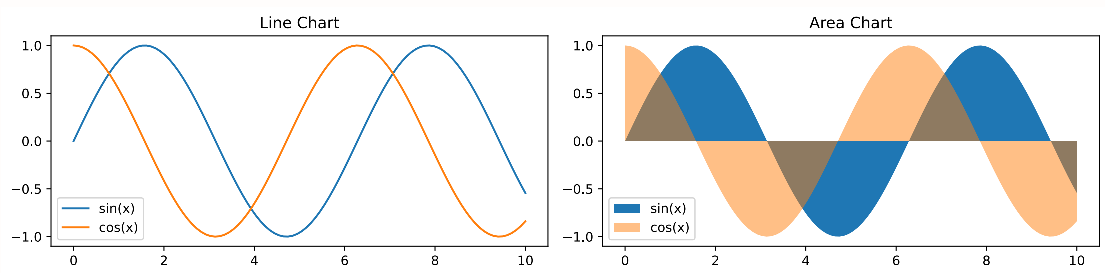


Line Charts and Area Charts

- Line charts: Show trends over time or continuous data
- Area charts: Similar to line charts, but with filled areas

```python
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)
```

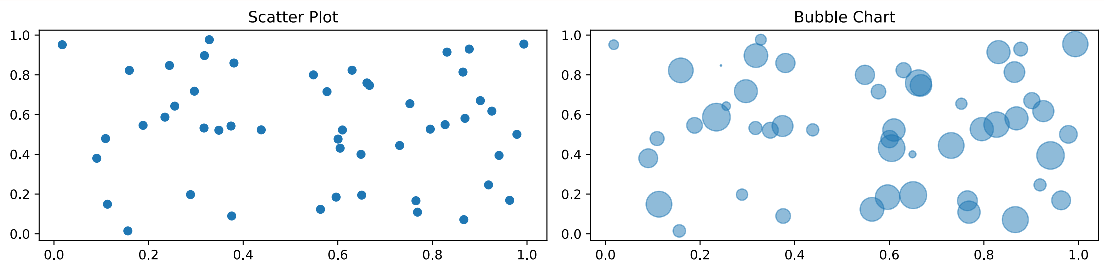
```
y1 = np.sin(x)
y2 = np.cos(x)

plt.figure(figsize=(12, 3))
plt.subplot(121)
plt.plot(x, y1, label='sin(x)')
plt.plot(x, y2, label='cos(x)')
plt.title('Line Chart')
plt.legend()

plt.subplot(122)
plt.fill_between(x, y1, label='sin(x)')
plt.fill_between(x, y2, label='cos(x)', alpha=0.5)
plt.title('Area Chart')
plt.legend()

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



Scatter Plots and Bubble Charts

- Scatter plots: Show relationship between two variables
- Bubble charts: Adds dimension with varying point sizes

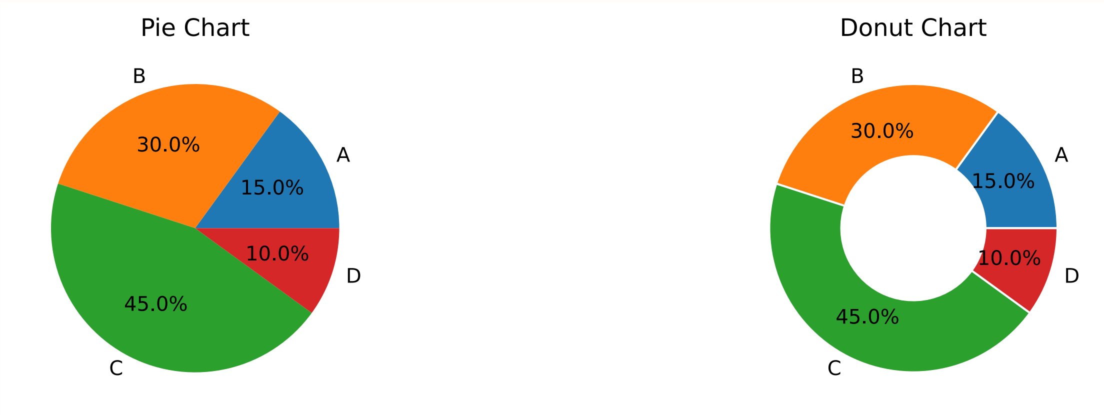
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.rand(50)
y = np.random.rand(50)
sizes = np.random.rand(50) * 500

plt.figure(figsize=(12, 3))
plt.subplot(121)
plt.scatter(x, y)
plt.title('Scatter Plot')

plt.subplot(122)
plt.scatter(x, y, s=sizes, alpha=0.5)
plt.title('Bubble Chart')

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



Pie Charts and Donut Charts

- Pie charts: Show composition of a whole
- Donut charts: Similar to pie charts, but with a hole

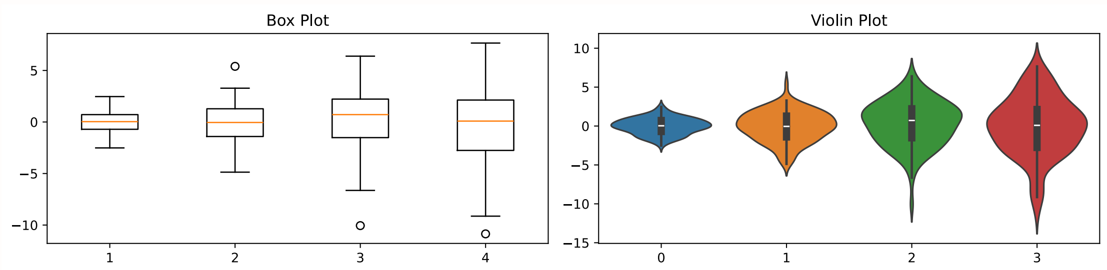
```python
import matplotlib.pyplot as plt

labels = 'A', 'B', 'C', 'D'
sizes = [15, 30, 45, 10]

plt.figure(figsize=(12, 3))

# Pie chart (left subplot)
plt.subplot(121)
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.title('Pie Chart')

# Donut chart (right subplot)
plt.subplot(122)
plt.pie(sizes, labels=labels, autopct='%1.1f%%',
        pctdistance=0.7, labeldistance=1.1,
        wedgeprops=dict(width=0.5, edgecolor='white'))
plt.title('Donut Chart')

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



Box Plots and Violin Plots

- Box plots: Show distribution of data through quartiles
- Violin plots: Combine box plot with kernel density

```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

data = [np.random.normal(0, std, 100) for std in range(1, 5)]

plt.figure(figsize=(12, 3))
plt.subplot(121)
plt.boxplot(data)
plt.title('Box Plot')

plt.subplot(122)
sns.violinplot(data)
plt.title('Violin Plot')

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

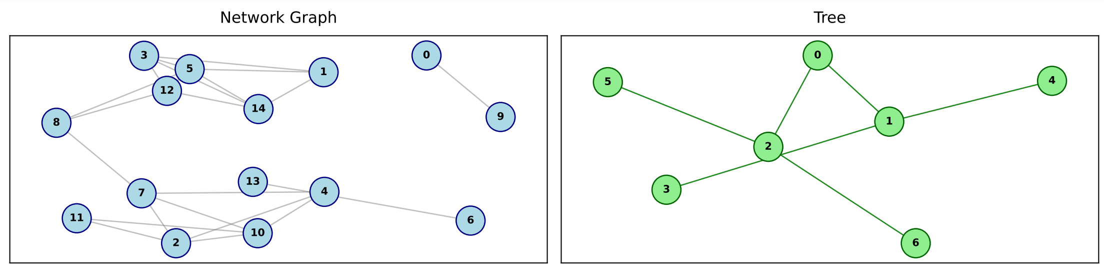


Network Graphs and Trees

- Network graphs: Show relationships between entities
- Tree diagrams: Display hierarchical structures

```python
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np

# Create figure
plt.figure(figsize=(12, 3))

# Network graph (left subplot)
plt.subplot(121)
G = nx.random_geometric_graph(15, 0.3)  # Reduced nodes for clarity
pos = nx.spring_layout(G, k=1, seed=42)  # Better layout with fixed seed
nx.draw_networkx_nodes(G, pos,
                      node_color='lightblue',
                      node_size=500,
                      edgecolors='navy',
                      linewidths=1)
nx.draw_networkx_edges(G, pos,
                      edge_color='gray',
                      width=1,
                      alpha=0.5)
nx.draw_networkx_labels(G, pos,
```

```
                           font_size=8,
                           font_weight='bold')
plt.title('Network Graph', pad=10)

# Tree diagram (right subplot)
plt.subplot(122)
T = nx.balanced_tree(2, 2)  # Create a balanced tree with 2 children, depth
2
pos_tree = nx.spring_layout(T, k=1.5, seed=42)
nx.draw_networkx_nodes(T, pos_tree,
                      node_color='lightgreen',
                      node_size=500,
                      edgecolors='darkgreen',
                      linewidths=1)
nx.draw_networkx_edges(T, pos_tree,
                      edge_color='forestgreen',
                      width=1)
nx.draw_networkx_labels(T, pos_tree,
                       font_size=8,
                       font_weight='bold')
plt.title('Tree', pad=10)

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



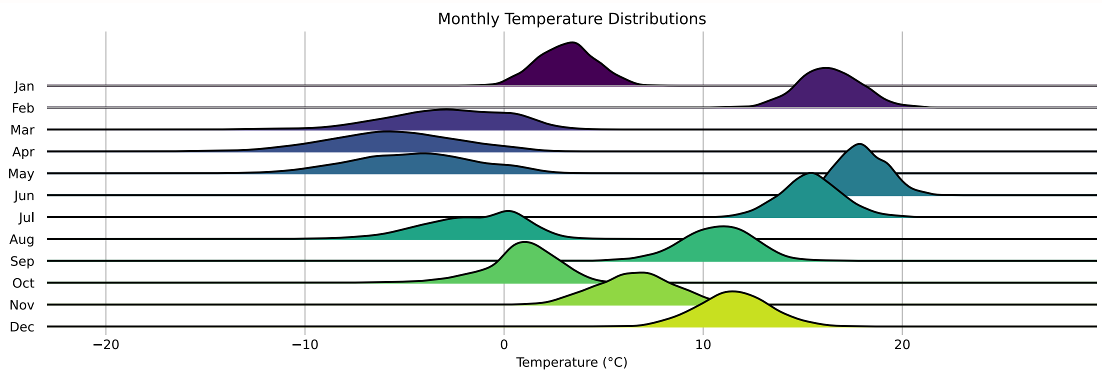
Ridgeline Plots

- Ridgeline plots: Show distribution of data across categories

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

# Create realistic temperature distributions
np.random.seed(42)
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
'Oct', 'Nov', 'Dec']
data = []

# Temperature parameters for Helsinki, Finland (as an example)
mean_temps = [
    -3.5,   # Jan
    -4.5,   # Feb
```

```
    -1.0,    # Mar
    4.5,     # Apr
    10.8,    # May
    15.5,    # Jun
    18.0,    # Jul
    16.3,    # Aug
    11.5,    # Sep
    6.6,     # Oct
    1.6,     # Nov
    -2.0     # Dec
]

# Winter months have more variance than summer months
variances = [
    2.5,     # Jan
    2.5,     # Feb
    2.2,     # Mar
    2.0,     # Apr
    1.8,     # May
    1.5,     # Jun
    1.2,     # Jul
    1.5,     # Aug
    1.8,     # Sep
    2.0,     # Oct
    2.2,     # Nov
    2.5      # Dec
]

for month, mean_temp, variance in zip(months, mean_temps, variances):
    # Add some random noise to make it more natural
    distribution = np.random.normal(loc=mean_temp, scale=variance,
size=1000)
    # Add slight skewness to winter months (more extreme cold than warm
days)
    if mean_temp < 5:
        distribution = distribution - 0.3 * np.abs(distribution)
    data.append(pd.DataFrame({
        'temperature': distribution,
        'month': month
    }))

df = pd.concat(data, ignore_index=True)

# Create the ridgeline plot
joypy.joyplot(
    data=df,
    by="month",
    column="temperature",
    colormap=plt.cm.viridis,
    title="Monthly Temperature Distributions",
    labels=months,
    range_style='all',
    tails=0.2,
    overlap=0.7,
```

```
    grid=True,
    figsize=(12, 4)
)

plt.xlabel("Temperature (°C)")
plt.show()
```



How to Plot in Python

Python Plotting Libraries

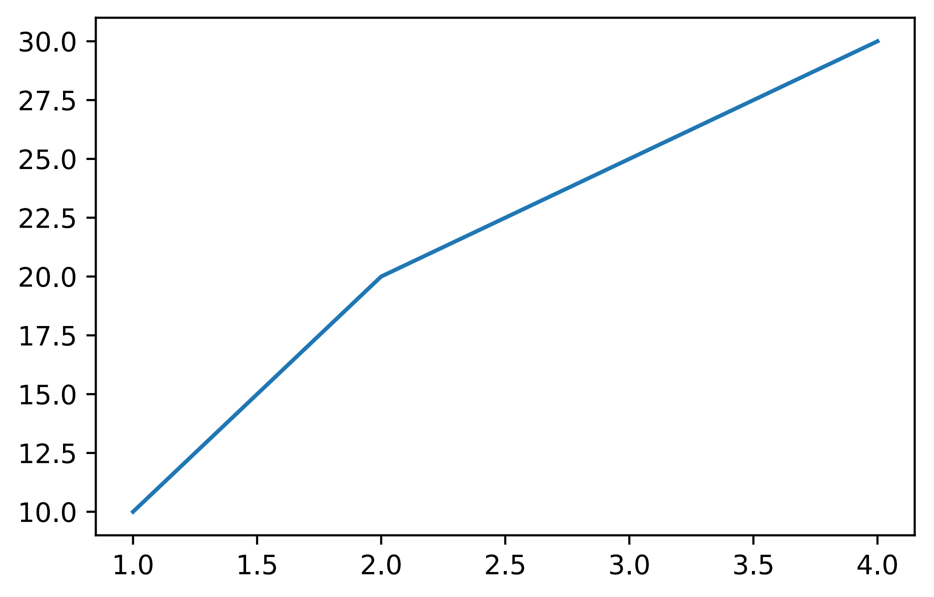
- There are many libraries for data visualization in Python
  - Matplotlib: The foundation for most Python plotting libraries
  - Seaborn: Interface for statistical data visualization
  - Plotly: Interactive and customizable plotting library
  - Bokeh: Interactive and complex plots
  - Joypy: Easy ridgeline plots

Matplotlib Module

- Matplotlib is the foundation for most Python plotting libraries
- Customizable and suitable for high-quality figures
- Provides easy to use functions for plotting
- Works well with Pandas DataFrames

Basic Matplotlib Example

```
import matplotlib.pyplot as plt # .pyplot is the main module in the package
plt.plot([1, 2, 3, 4], [10, 20, 25, 30]) # first is x-axis, second is y-
axis
plt.show()
```

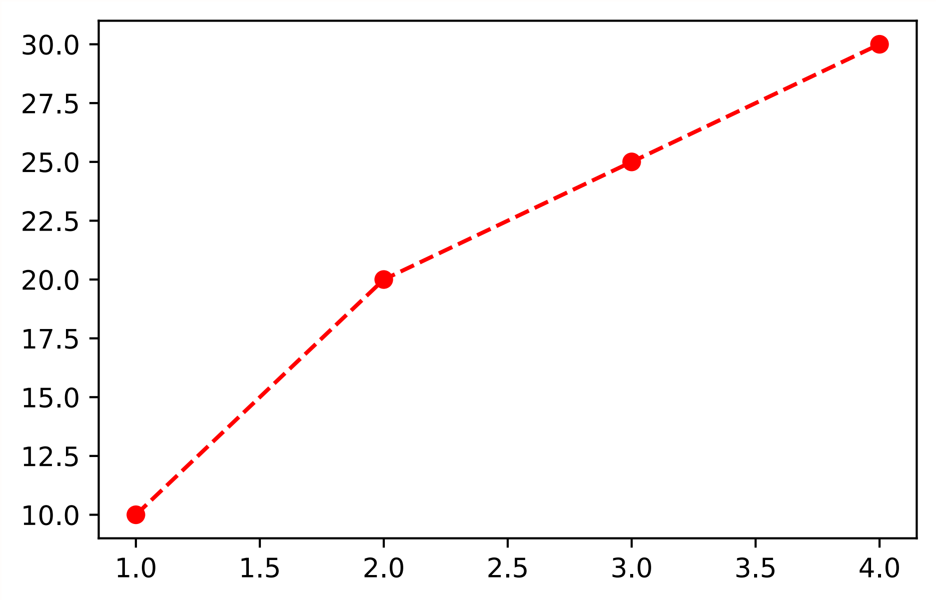


Customizing Plots

- Line Types: with `ls=`
  - `-`, `--`, `-.`, `:`, `None`;
- Colors: with `color=`
  - `r`, `g`, `b`, `c`, `m`, `y`, `k`, ...
- Markers: with `marker=`
  - `o`, `s`, `D`, `p`, `*`, `x`, ...
- Labels: with `label=`, `title=`, `xlabel=`, `ylabel=`

Red dashed line with circles

```
plt.plot(
    [1, 2, 3, 4],
    [10, 20, 25, 30],
    color='red',
    linestyle='--',
    marker='o')
plt.show()
```

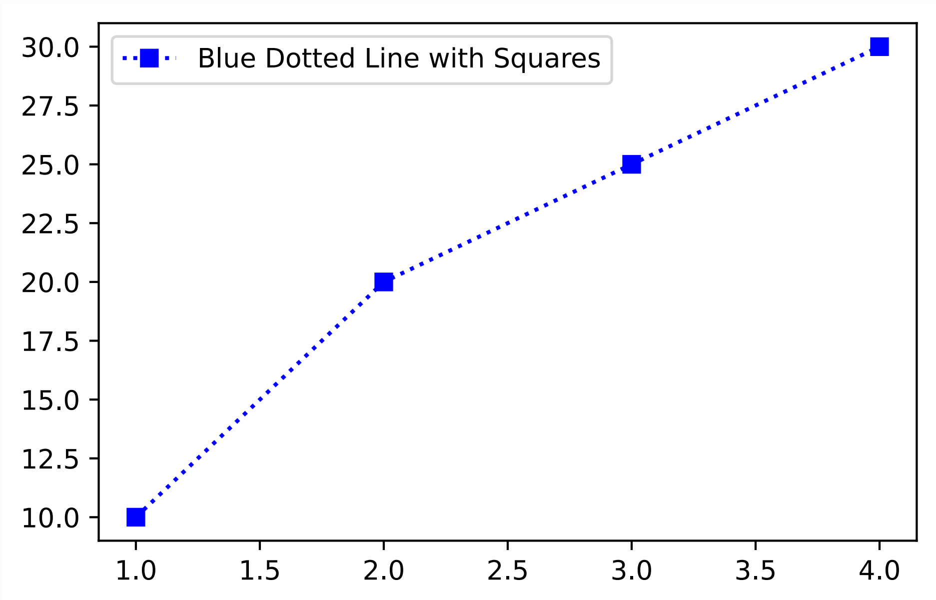


Blue dotted line with squares

Question: How can we create such a plot?

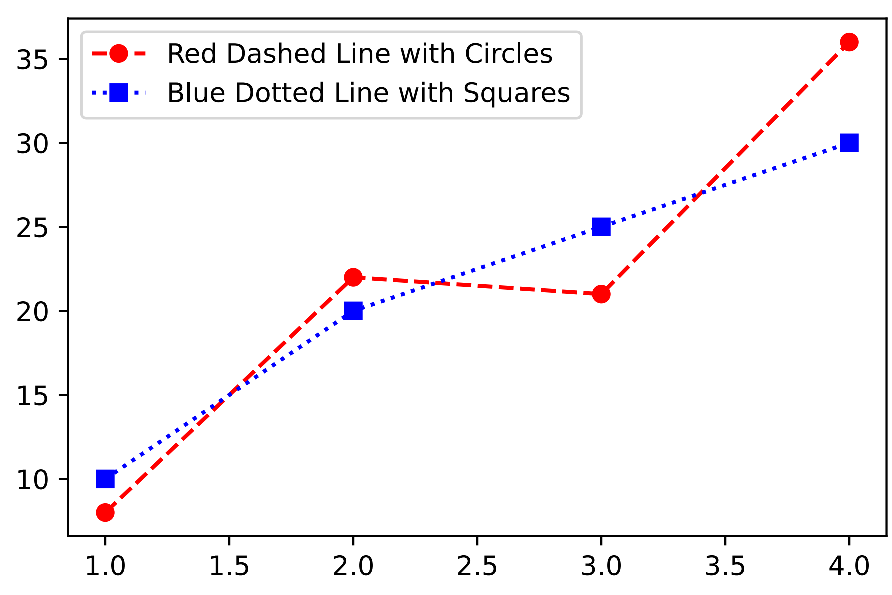
...

```
plt.plot(
    [1, 2, 3, 4],
    [10, 20, 25, 30],
    color='blue',
    linestyle=':',
    marker='s',
    label='Blue Dotted Line with Squares')
plt.legend()
plt.show()
```



Multiple Plots

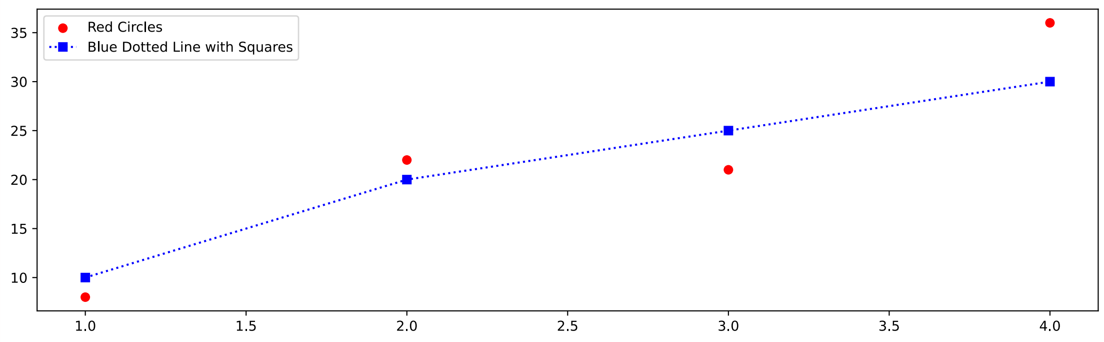
```
plt.plot(
    [1, 2, 3, 4],
    [8, 22, 21, 36],
    color='red',
    linestyle='--',
    marker='o',
    label='Red Dashed Line with Circles')
plt.plot(
    [1, 2, 3, 4],
    [10, 20, 25, 30],
    color='blue',
    linestyle=':',
    marker='s',
    label='Blue Dotted Line with Squares')
plt.legend()
plt.show()
```



Type and Size of the Plot

```
plt.figure(figsize=(14, 4))
plt.scatter(
    [1, 2, 3, 4],
    [8, 22, 21, 36],
    color='red',
    marker='o',
    label='Red Circles')
plt.plot(
    [1, 2, 3, 4],
    [10, 20, 25, 30],
    color='blue',
    linestyle=':',
    marker='s',
```

```
    label='Blue Dotted Line with Squares')
plt.legend()
plt.show()
```

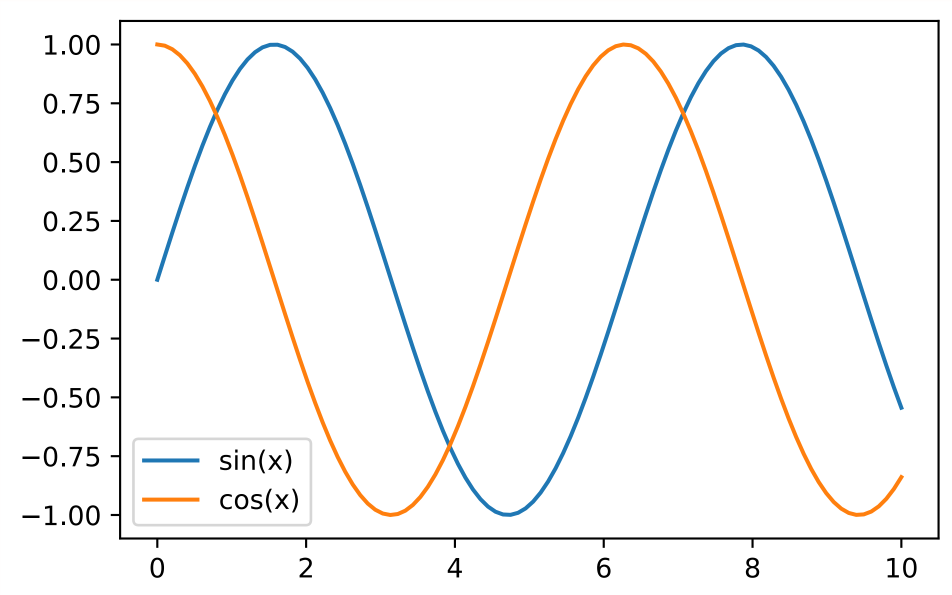


Plots in Action

Task: Create two line plots of the following data:

```
# Make sure to label the plots! Color and marker are optional.
import numpy as np

x = np.linspace(0, 10, 100) # 100 points between 0 and 10
y1 = np.sin(x) # sine function
y2 = np.cos(x) # cosine function
```

Solution



Plotting the Right Way

The Message Matters

- Making beautiful plots is rather easy[1]

- It is important to understand the underlying data
- What kind of plots are appropriate for your data?
- What is the message you want to convey?

Temperature Dataset

Let's load the temperature dataset from our last tutorial.

```
import pandas as pd
df = pd.read_excel('supplementary/lec_09/temp_anomaly_data.xlsx')
print(df.head())
```

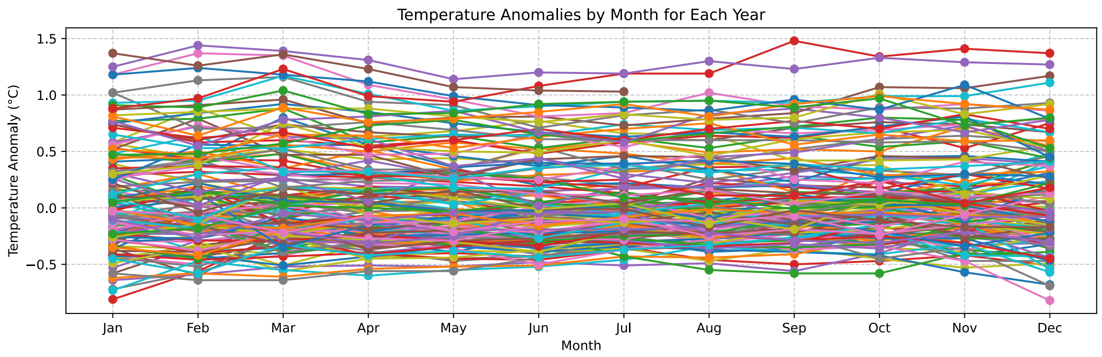
```
   Year Month  Anomaly
0  1880   Jan    -0.18
1  1881   Jan    -0.20
2  1882   Jan     0.16
3  1883   Jan    -0.29
4  1884   Jan    -0.13
```

Example of a bad plot

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

# Convert Month to numeric for proper ordering
month_map = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6,
             'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}
df['Month_num'] = df['Month'].map(month_map)

# Sort by Year and Month
df = df.sort_values(['Year', 'Month_num'])

# Create the plot
plt.figure(figsize=(12, 4))

# Plot each year as a separate line
for year in df['Year'].unique():
    year_data = df[df['Year'] == year]
    plt.plot(year_data['Month_num'], year_data['Anomaly'], label=str(year),
marker='o')

# Customize the plot
plt.title('Temperature Anomalies by Month for Each Year')
plt.xlabel('Month')
plt.ylabel('Temperature Anomaly (°C)')
plt.grid(True, linestyle='--', alpha=0.7)

# Set x-axis ticks to show month names
plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                          'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
```

[1] At least nowadays with the assistance of AI!

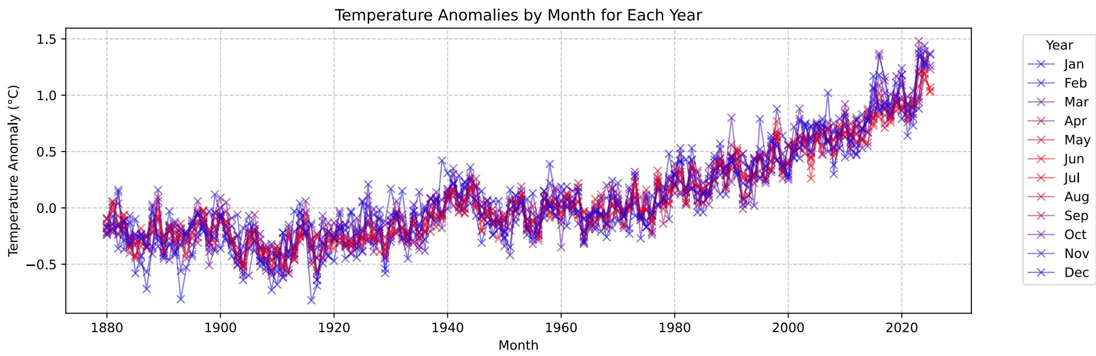
```
# Adjust layout to prevent legend cutoff
plt.tight_layout()

plt.show()
```



An okay plot

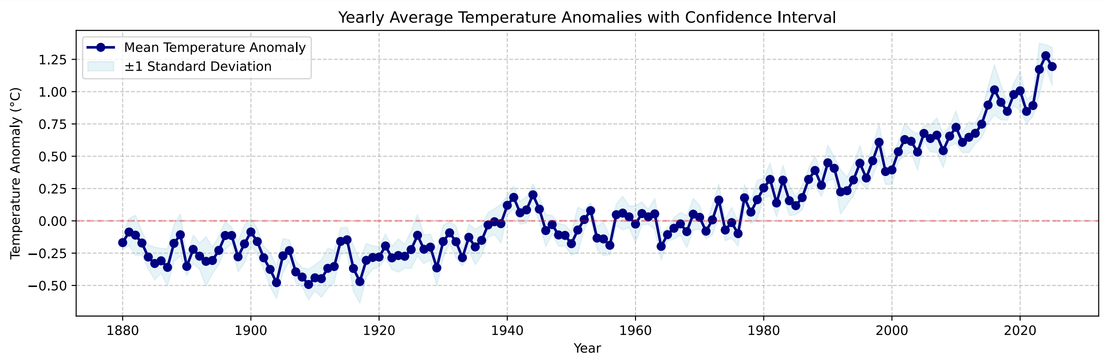
```
import pandas as pd
import matplotlib.pyplot as plt

# Define correct month order
month_order = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
               'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

# Pivot the data and reorder columns
pivot_df = df.pivot(index='Year', columns='Month', values='Anomaly')
pivot_df = pivot_df[month_order]  # Reorder columns according to
month_order

# Create the plot with a blue-to-red gradient for winter-to-summer
fig, ax = plt.subplots(figsize=(12, 4))  # Create figure and axes objects

# Create color gradient
colors = []
for i in range(12):
    if i <= 5:  # January to June
        r = i / 5
        b = 1 - (i / 5)
        colors.append((r, 0, b))
    else:  # July to December
        r = 1 - ((i-6) / 5)
        b = (i-6) / 5
        colors.append((r, 0, b))

pivot_df.plot(ax=ax, marker='x', linewidth=1, alpha=0.5, color=colors)

plt.title('Temperature Anomalies by Month for Each Year')
plt.xlabel('Month')
plt.ylabel('Temperature Anomaly (°C)')
plt.grid(True, linestyle='--', alpha=0.7)
```

```python
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', title='Year')
plt.tight_layout()
plt.show()
```



A better plot

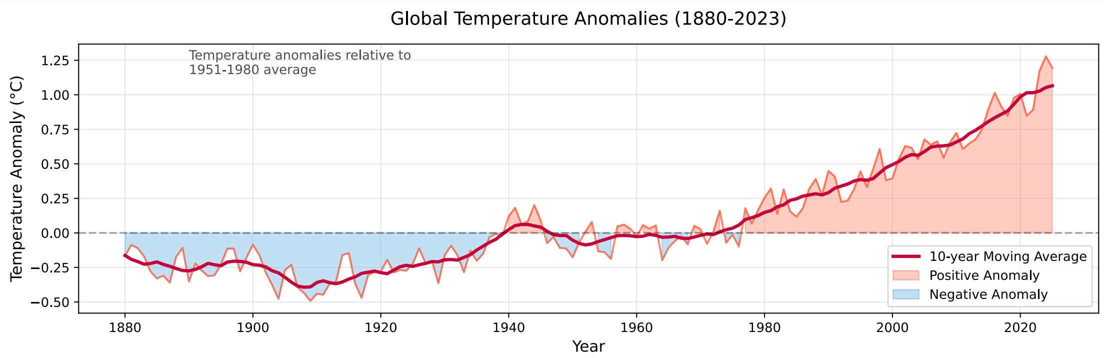
```python
# Calculate yearly averages
yearly_means = df.groupby('Year')['Anomaly'].agg(['mean', 'std'])

# Create the plot
plt.figure(figsize=(12, 4))

# Plot mean values as a line
plt.plot(yearly_means.index, yearly_means['mean'],
         color='navy', linewidth=2, marker='o',
         label='Mean Temperature Anomaly')

# Add shaded area for standard deviation
plt.fill_between(yearly_means.index,
                 yearly_means['mean'] - yearly_means['std'],
                 yearly_means['mean'] + yearly_means['std'],
                 color='lightblue', alpha=0.3,
                 label='±1 Standard Deviation')

# Customize the plot
plt.title('Yearly Average Temperature Anomalies with Confidence Interval')
plt.xlabel('Year')
plt.ylabel('Temperature Anomaly (°C)')
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()

# Add zero reference line
plt.axhline(y=0, color='red', linestyle='--', alpha=0.3)

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



A good plot

```
# Read the data
df = pd.read_excel('supplementary/lec_09/temp_anomaly_data.xlsx')

# Aggregate to yearly averages
yearly_df = df.groupby('Year')['Anomaly'].mean().reset_index()

# Set the style
fig, ax = plt.subplots(figsize=(12, 4))

# Create the main line plot
plt.plot(yearly_df['Year'], yearly_df['Anomaly'],
         color='#FF5733',
         linewidth=1.5,
         alpha=0.7)

# Calculate rolling mean on yearly data
rolling_mean = yearly_df['Anomaly'].rolling(window=10, center=True,
min_periods=5).mean()
plt.plot(yearly_df['Year'], rolling_mean,
         color='#C70039',
         linewidth=2.5,
         label='10-year Moving Average')

# Fill between the line and zero
plt.fill_between(yearly_df['Year'], yearly_df['Anomaly'], 0,
                 where=(yearly_df['Anomaly'] >= 0),
                 color='#FF5733',
                 alpha=0.3,
                 label='Positive Anomaly')
plt.fill_between(yearly_df['Year'], yearly_df['Anomaly'], 0,
                 where=(yearly_df['Anomaly'] < 0),
                 color='#3498DB',
                 alpha=0.3,
                 label='Negative Anomaly')

# Customize the plot
plt.title('Global Temperature Anomalies (1880-2023)',
         fontsize=14,
         pad=15)
```

```python
plt.xlabel('Year', fontsize=12)
plt.ylabel('Temperature Anomaly (°C)', fontsize=12)
plt.grid(True, alpha=0.3)
plt.legend()

# Add a horizontal line at y=0
plt.axhline(y=0, color='black', linestyle='--', alpha=0.3)

# Add text annotation for context
plt.text(1890, 1.15,
         'Temperature anomalies relative to\n1951-1980 average',
         fontsize=10,
         alpha=0.7)

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



How to build such a plot?

- Think: about what you want to build
- Describe: what you want to build in detail
- Use AI: to build the plot for you
- Use Libraries: documentation to fine-tune the plot

...

> **Tip**
>
> As usual, the best way to learn is by doing! AI makes it very easy to get started.

Good Plotting in Action

Task: Create a plot of your own for the data.

...

```python
# TODO: Load the data from the `temp_anomaly_data.xlsx` file you have saved
last lecture yourself and plot the temperature anomaly data. Find a way to
make the plot meaningful and attractive in order to tell a story for the
```

```
reader.
# YOUR CODE HERE
```

Creating Dashboards

Dash for Dashboards

- Dash is a framework for building web applications
- It is built on top of Flask, Plotly.js, and React.js
- This lecture is build on top of React.js
- It is very customizable and has a lot of examples

Panel for Dashboards

- Panel is built on top of Bokeh (instead of Plotly.js)
- Reasonably easy to use, but not super easy
- Highly customizable, also for multiple pages
- Good performance even for more complex dashboards

Streamlit for Dashboards

- Streamlit is a rather new and popular library
- Very easy and fast way to build dashboards
- Performance only good on simpler dashboards
- Not as many examples and as customizable

NiceGUI for Dashboards

- NiceGUI is also a relatively new library
- Very customizable and a large fan base
- Not as many examples and as easy to use as Streamlit
- Allows building web-based desktop applications

Which one to choose?

- Streamlit: if you want to build a dashboard fast
- Dash: if you want more flexibility and Plotly.js
- Panel: if you want a Bokeh-based solution with more flexibility
- NiceGUI: if you want to build a desktop-like application

. . .

> Tip
>
> There are many opinions on which tool is the best one. My approach is usually just to try the main contenders to see which one suits my workflow best.

How to build a dashboard

- We won't go into details on how to build dashboards
- The best way to learn is by doing!

- Use AI and the libraries documentation as starting points
- We will also build a dashboard in today's tutorial

Creating GUIs

PySide6 for GUIs

- Here, the recommendation is much easier!
- In the past, `tkinter` was often the way to go
- But currently, my recommendation is `PySide6`
- It's a mature library we can use to build cross-platform desktop applications

. . .

> **ℹ Note**
>
> And that's it for todays lecture!
> You now have the basic knowledge to start working with Plots, Dashboards and GUIs!.

Literature

Interesting Books

- Wilke, C. (2019). Fundamentals of data visualization: A primer on making informative and compelling figures (First edition). O'Reilly Media.
  - A book that is highly recommended to understand the principles of data visualization and how to create effective visualizations.
  - Link to the free book website

. . .

For more interesting literature to learn more about Python, take a look at the literature list of this course.