CE49X: Introduction to Computational Thinking and Data Science for Civil Engineers

Week 5: Matplotlib and Seaborn Visualization

Dr. Eyuphan Koc

Bogazici University

Fall 2025

Course Outline

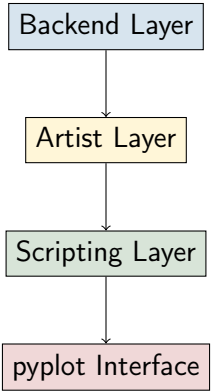
What is Matplotlib?

- **Multi-platform data visualization library** built on NumPy arrays
- Designed to work with the broader SciPy stack
- Created by John Hunter in 2002 as Python alternative to MATLAB
- **Cross-platform**: Works on many operating systems and graphics backends
- **Foundation** for many other visualization tools (Seaborn, Pandas plotting, etc.)

Key Features

- Professional-quality plots for publications
- Interactive plotting capabilities
- Extensive customization options
- Integration with Jupyter notebooks

Matplotlib Architecture



- **Backend**: Handles rendering (PNG, PDF, SVG, interactive windows)
- **Artist**: Object-oriented interface for fine control
- **Scripting**: Convenient functions for common tasks
- **pyplot**: MATLAB-like interface (most commonly used)

Basic Import and Setup

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

# For Jupyter notebooks
%matplotlib inline

# Set style
plt.style.use('seaborn-whitegrid')
```

Two Interfaces

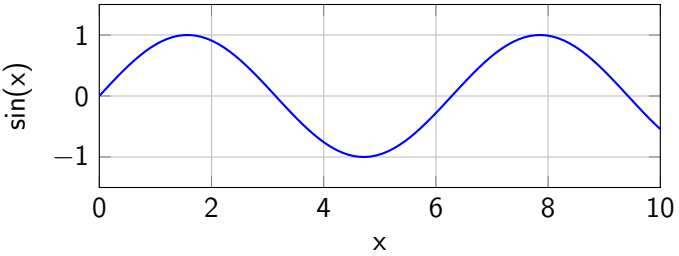
- **MATLAB-style**: `plt.plot()`, `plt.xlabel()`, etc.
- **Object-oriented**: `fig, ax = plt.subplots()`, `ax.plot()`

Best Practice

Use object-oriented interface for complex plots, MATLAB-style for simple plots

Creating Simple Line Plots

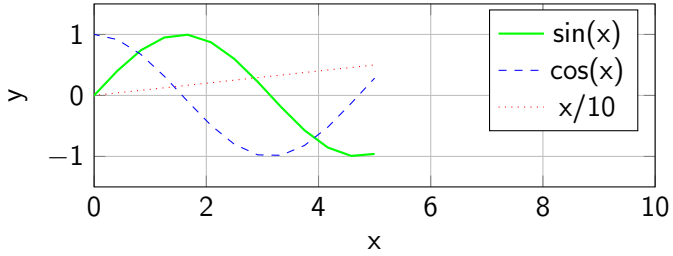
```python
# Object-oriented approach
fig, ax = plt.subplots()
x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x))

# MATLAB-style approach
plt.plot(x, np.sin(x))
```



Multiple Lines and Styling

```
# Multiple lines
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.legend()

# Line styles
plt.plot(x, x + 0, linestyle='solid')    # '-'
plt.plot(x, x + 1, linestyle='dashed')  # '--'
plt.plot(x, x + 2, linestyle='dashdot') # '-.'
plt.plot(x, x + 3, linestyle='dotted')   # ':'
```

Multiple Lines Output



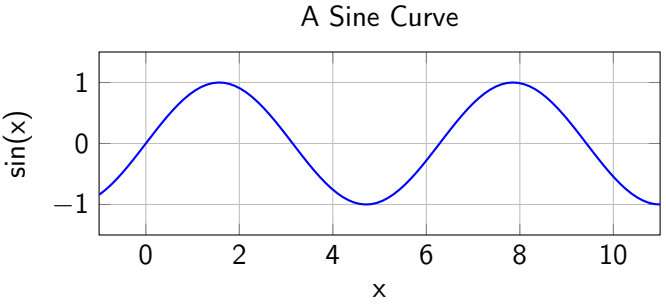
Color Options

- **Named colors**: 'blue', 'red', 'green'
- **Short codes**: 'b', 'r', 'g', 'c', 'm', 'y', 'k'
- **Hex codes**: '#FFDD44'
- **RGB tuples**: (1.0, 0.2, 0.3)

Axis Control and Labels

```
plt.plot(x, np.sin(x))

# Axis limits
plt.xlim(-1, 11)
plt.ylim(-1.5, 1.5)

# Alternative: plt.axis([-1, 11, -1.5, 1.5])

# Labels and title
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.title('A Sine Curve')

# Object-oriented equivalent
ax.set_xlim(-1, 11)
ax.set_ylim(-1.5, 1.5)
ax.set_xlabel('x')
ax.set_ylabel('sin(x)')
ax.set_title('A Sine Curve')
```

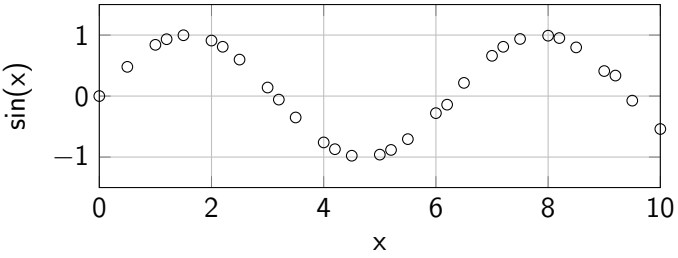
Axis Control Output



Basic Scatter Plots

```
# Using plt.plot with markers
x = np.linspace(0, 10, 30)
y = np.sin(x)
plt.plot(x, y, 'o', color='black')

# Using plt.scatter
plt.scatter(x, y, marker='o')
```

Scatter Plot Output



Marker Styles

- **Basic**: 'o', '.', ',', 'x', '+'
- **Triangles**: 'v', '^', '<', '>'
- **Squares**: 's', 'd'
- **Stars**: '*', 'p', 'h'

Advanced Scatter Plots

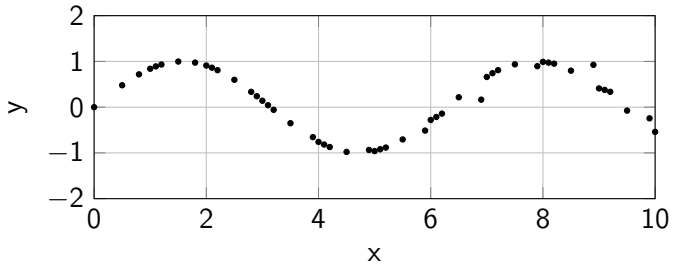
```
# Variable size and color
rng = np.random.RandomState(0)
x = rng.randn(100)
y = rng.randn(100)
colors = rng.rand(100)
sizes = 1000 * rng.rand(100)

plt.scatter(x, y, c=colors, s=sizes,
            alpha=0.3, cmap='viridis')
plt.colorbar()
```

plt.plot vs plt.scatter

- **plt.plot**: Faster for large datasets (>1000 points)
- **plt.scatter**: More flexible for variable properties

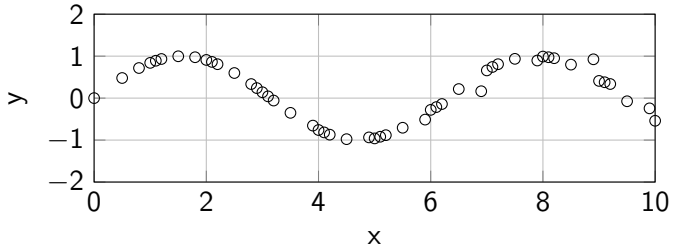
Basic Error Bars

```
x = np.linspace(0, 10, 50)
dy = 0.8
y = np.sin(x) + dy * np.random.randn(50)

plt.errorbar(x, y, yerr=dy, fmt='.k')
```




```
# Customized error bars
plt.errorbar(x, y, yerr=dy, fmt='o', color='black',
             ecolor='lightgray', elinewidth=3, capsize=0)
```

Customized Error Bars Output

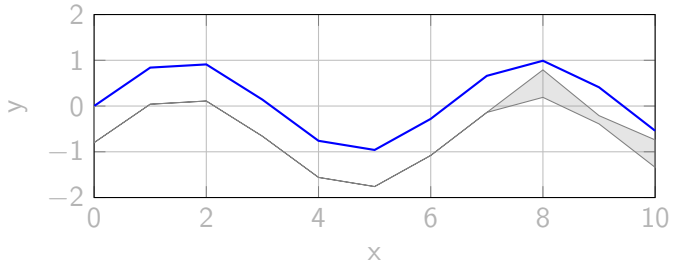


Error Bar Options

- **yerr**: Vertical error bars
- **xerr**: Horizontal error bars
- **fmt**: Format string for points
- **ecolor**: Error bar color
- **capsize**: Cap size

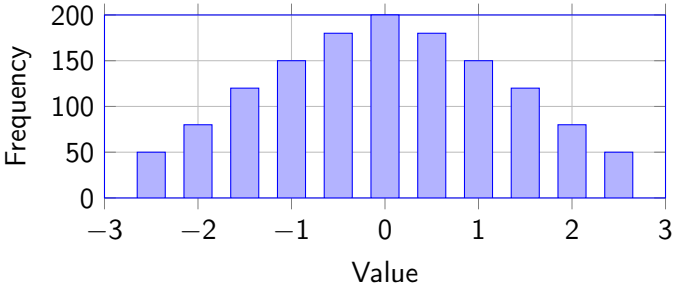
Continuous Error Regions

```
# Using fill_between for continuous errors
plt.plot(xdata, ydata, 'or')
plt.plot(xfit, yfit, '-', color='gray')

plt.fill_between(xfit, yfit - dyfit, yfit + dyfit,
                 color='gray', alpha=0.2)
```



Basic Histograms

```
data = np.random.randn(1000)

# Basic histogram
plt.hist(data)

# Customized histogram
plt.hist(data, bins=30, alpha=0.5,
         histtype='stepfilled', color='steelblue',
         edgecolor='none')
```



Multiple Histograms

```
x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)

kwargs = dict(histtype='stepfilled', alpha=0.3,
              normed=True, bins=40)

plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs)
```

2D Histograms

- **plt.hist2d()**: 2D histogram
- **plt.hexbin()**: Hexagonal binning
- **Kernel Density Estimation**: Smooth density estimation

Contour Plots

```
def f(x, y):
    return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)

x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)

# Line contours
plt.contour(X, Y, Z, colors='black')

# Filled contours
plt.contourf(X, Y, Z, 20, cmap='RdGy')
plt.colorbar()
```

Contour Functions

- **plt.contour()**: Line contours
- **plt.contourf()**: Filled contours
- **plt.imshow()**: Image representation

Combined Contour and Image Plots

```
# Combine contours with image
contours = plt.contour(X, Y, Z, 3, colors='black')
plt.clabel(contours, inline=True, fontsize=8)

plt.imshow(Z, extent=[0, 5, 0, 5], origin='lower',
           cmap='RdGy', alpha=0.5)
plt.colorbar()
```

imshow Gotchas

- Must specify **extent** manually
- **origin='lower'** for mathematical plots
- **aspect='image'** for equal scaling

Creating Subplots

```
# Manual subplot positioning
fig = plt.figure()
ax1 = fig.add_axes([0.1, 0.5, 0.8, 0.4])
ax2 = fig.add_axes([0.1, 0.1, 0.8, 0.4])

# Simple grid
for i in range(1, 7):
    plt.subplot(2, 3, i)
    plt.text(0.5, 0.5, str((2, 3, i)), ha='center')
```

Subplot Methods

- **plt.subplot(r, c, i)**: Single subplot
- **plt.subplots(r, c)**: Grid of subplots
- **plt.GridSpec()**: Flexible layouts

Advanced Subplot Layouts

```
# Using subplots with shared axes
fig, ax = plt.subplots(2, 3, sharex='col', sharey='row')

# Using GridSpec for complex layouts
grid = plt.GridSpec(2, 3, wspace=0.4, hspace=0.3)
plt.subplot(grid[0, 0])
plt.subplot(grid[0, 1:])
plt.subplot(grid[1, :2])
plt.subplot(grid[1, 2])
```

GridSpec Advantages

- **Flexible positioning**: Subplots can span multiple cells
- **Complex layouts**: Easy to create sophisticated arrangements
- **Consistent spacing**: Automatic spacing control

Available Stylesheets

```
# List available styles
print(plt.style.available)

# Use a style
plt.style.use('seaborn-whitegrid')

# Temporary style context
with plt.style.context('ggplot'):
    plt.plot(x, y)
```

Popular Styles

- **default**: Matplotlib default
- **seaborn-whitegrid**: Clean grid background
- **ggplot**: R ggplot2 style
- **fivethirtyeight**: FiveThirtyEight style
- **dark_background**: Dark theme

Customizing rcParams

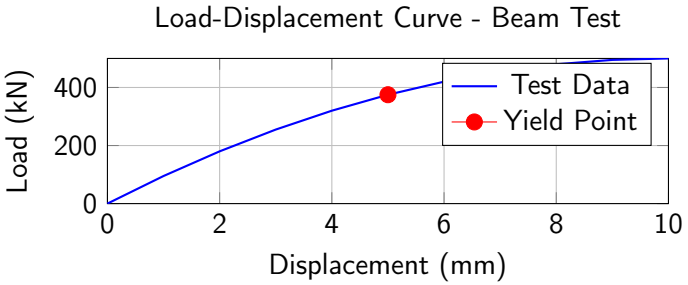
```
# Customize global settings
plt.rc('axes', facecolor='#E6E6E6', grid=True)
plt.rc('grid', color='w', linestyle='solid')
plt.rc('xtick', direction='out', color='gray')
plt.rc('ytick', direction='out', color='gray')
plt.rc('lines', linewidth=2)

# Reset to defaults
plt.rcParams.update(plt.rcParamsDefault)
```

Best Practices

- Use **stylesheets** for consistent themes
- Modify **rcParams** for global changes
- Keep **individual plot** customizations minimal

Structural Analysis Visualization

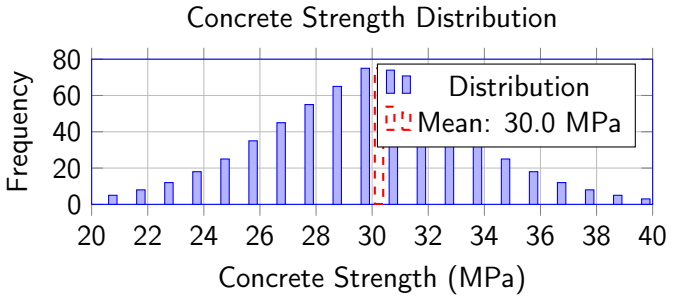
```
displacement = np.linspace(0, 10, 50)
load = 100 * displacement - 5 * displacement**2

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(displacement, load, 'b-', linewidth=2,
        label='Test Data')
ax.set_xlabel('Displacement (mm)')
ax.set_ylabel('Load (kN)')
ax.set_title('Load-Displacement Curve - Beam Test')
ax.grid(True, alpha=0.3)
ax.legend()
```



Material Property Analysis

```
# Concrete strength distribution
strengths = np.random.normal(30, 3, 1000)

plt.hist(strengths, bins=30, alpha=0.7,
         color='steelblue', edgecolor='black')
plt.axvline(np.mean(strengths), color='red',
            linestyle='--', label=f'Mean: {np.mean(strengths):.1f} MPa')
plt.xlabel('Concrete Strength (MPa)')
plt.ylabel('Frequency')
plt.title('Concrete Strength Distribution')
plt.legend()
```

Material Property Analysis Output



Geotechnical Data Visualization

```
# Soil layer visualization
depths = [0, 2, 5, 8, 12]
shear_strengths = [20, 35, 45, 60, 80]

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Shear strength profile
ax1.plot(shear_strengths, depths, 'o-', linewidth=2)
ax1.set_xlabel('Shear Strength (kPa)')
ax1.set_ylabel('Depth (m)')
ax1.set_title('Shear Strength Profile')
ax1.invert_yaxis()

# Contour plot of stress distribution
x = np.linspace(0, 10, 50)
y = np.linspace(0, 15, 50)
X, Y = np.meshgrid(x, y)
stress = 100 * np.exp(-Y/5) * np.sin(X/2)

contour = ax2.contourf(X, Y, stress, levels=20, cmap='viridis')
ax2.set_xlabel('Distance (m)')
ax2.set_ylabel('Depth (m)')
ax2.set_title('Stress Distribution')
plt.colorbar(contour, ax=ax2, label='Stress (kPa)')
```

Professional Plot Guidelines

Design Principles

- **Clarity**: Clear labels, appropriate font sizes
- **Consistency**: Use consistent colors and styles
- **Simplicity**: Avoid unnecessary decorations
- **Accessibility**: Consider colorblind-friendly palettes

Common Mistakes

- **Tiny fonts**: Use readable font sizes (12pt+)
- **Cluttered plots**: Remove unnecessary elements
- **Poor color choices**: Use high contrast colors
- **Missing labels**: Always label axes and include units

Saving and Exporting

```
# Save in different formats
fig.savefig('plot.png', dpi=300, bbox_inches='tight')
fig.savefig('plot.pdf', bbox_inches='tight')
fig.savefig('plot.svg', bbox_inches='tight')

# High-resolution for publications
fig.savefig('publication_plot.png',
            dpi=300, bbox_inches='tight',
            facecolor='white', edgecolor='none')
```

File Format Guidelines

- **PNG**: Web, presentations (lossless)
- **PDF**: Publications, vector graphics
- **SVG**: Web, scalable graphics
- **EPS**: LaTeX documents

Seaborn Statistical Visualization - What is Seaborn?

- **High-level statistical visualization library** built on Matplotlib
- Designed to work seamlessly with Pandas DataFrames
- Provides **sane defaults** for plot style and colors
- Created to address Matplotlib's limitations for statistical plots
- **Integration**: Works perfectly with NumPy, Pandas, and SciPy

Key Advantages

- **Statistical focus**: Built for data exploration and analysis
- **Beautiful defaults**: Modern, publication-ready aesthetics
- **DataFrame integration**: Automatic handling of labeled data
- **High-level functions**: Complex plots with simple commands

Why Use Seaborn?

Matplotlib Limitations

- **Outdated defaults**: Based on MATLAB circa 1999
- **Low-level API**: Requires lots of boilerplate code
- **No DataFrame integration**: Must extract Series manually
- **Poor statistical plots**: Not designed for data analysis

Seaborn Solutions

- **Modern aesthetics**: Beautiful, publication-ready defaults
- **High-level functions**: Statistical plots with simple commands
- **Pandas integration**: Automatic DataFrame handling
- **Statistical focus**: Built specifically for data exploration

Matplotlib vs Seaborn

Matplotlib

- **Low-level control**
- **Publication plots**
- **Custom graphics**
- **Scientific visualization**
- **Animation support**

Seaborn

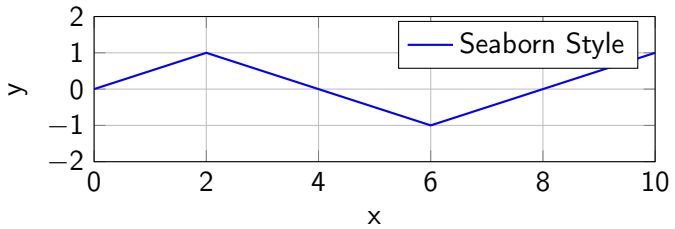
- **Statistical plots**
- **DataFrame integration**
- **Beautiful defaults**
- **Quick exploration**
- **Correlation analysis**

Best Practice

- **Use Seaborn** for statistical data exploration
- **Use Matplotlib** for custom, publication-quality plots
- **Combine both** for comprehensive visualization workflows

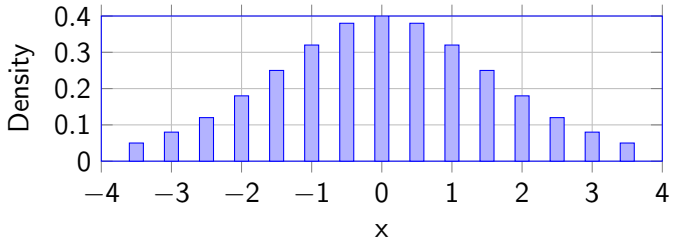
Seaborn vs Matplotlib Comparison

```python
# Matplotlib approach
import matplotlib.pyplot as plt
plt.style.use('classic')
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left')

# Seaborn approach
import seaborn as sns
sns.set()
plt.plot(x, y)  # Same code, better output!
plt.legend('ABCDEF', ncol=2, loc='upper left')
```



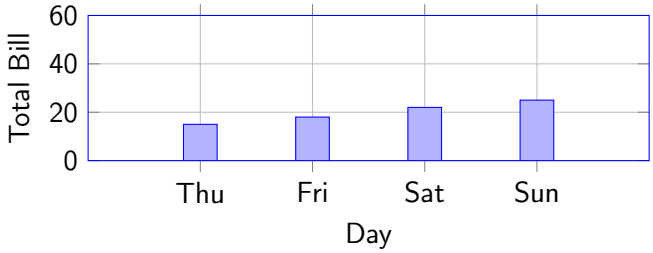
Statistical Distribution Plots

```
# Generate sample data
data = np.random.multivariate_normal([0, 0],
                                     [[5, 2], [2, 2]],
                                     size=2000)
data = pd.DataFrame(data, columns=['x', 'y'])

# Histogram with KDE
sns.distplot(data['x'], kde=True, rug=True)

# Joint distribution plot
sns.jointplot("x", "y", data, kind='kde')
```



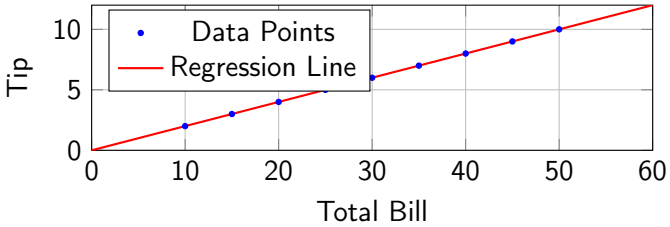
Categorical Data Visualization

```
# Load sample dataset
tips = sns.load_dataset('tips')

# Box plots
sns.boxplot(x="day", y="total_bill", data=tips)

# Violin plots
sns.violinplot(x="day", y="total_bill", data=tips)
```



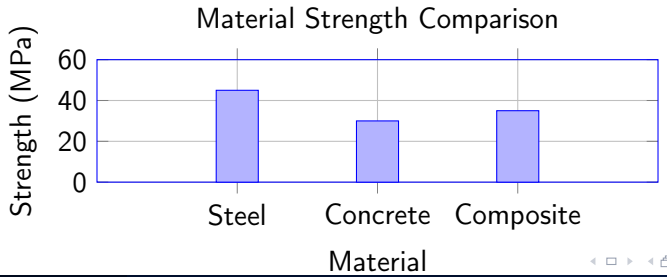
Regression and Correlation Plots

```
# Scatter plot with regression line
sns.regplot(x="total_bill", y="tip", data=tips)

# Correlation heatmap
corr = tips.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
```



Civil Engineering Applications

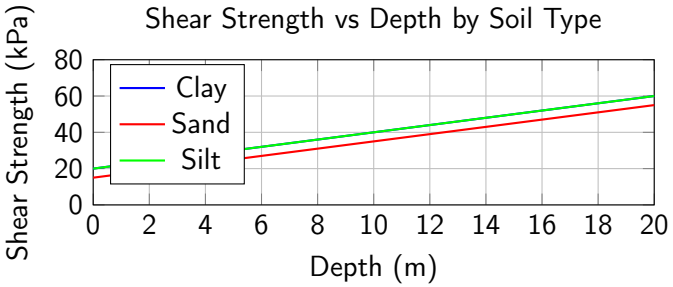
```
# Structural test data
np.random.seed(42)
beam_types = ['Steel', 'Concrete', 'Composite']
materials = np.random.choice(beam_types, 200)
strengths = np.random.normal(30, 5, 200) + \
            np.where(materials == 'Steel', 20,
                    np.where(materials == 'Composite', 10, 0))

df = pd.DataFrame({'Material': materials, 'Strength': strengths})

# Material comparison
sns.boxplot(x='Material', y='Strength', data=df)
plt.title('Material Strength Comparison')
```



Geotechnical Data Analysis

```
# Soil test data
depths = np.random.uniform(0, 20, 100)
soil_types = np.random.choice(['Clay', 'Sand', 'Silt'], 100)
shear_strength = 20 + 2*depths + \
                 np.random.normal(0, 5, 100) + \
                 np.where(soil_types == 'Clay', 10,
                         np.where(soil_types == 'Sand', -5, 0))

geo_df = pd.DataFrame({
    'Depth': depths,
    'Soil_Type': soil_types,
    'Shear_Strength': shear_strength
})

# Regression analysis
sns.lmplot(x='Depth', y='Shear_Strength',
           hue='Soil_Type', data=geo_df)
```

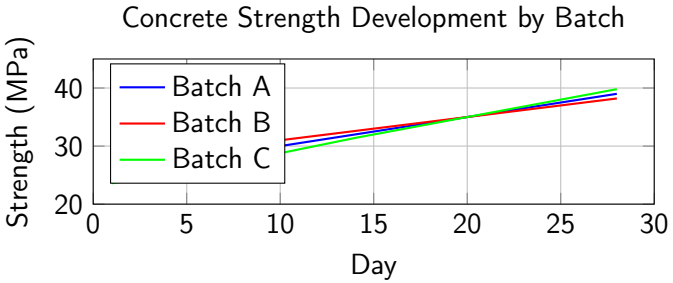
Geotechnical Analysis Output



Construction Quality Control

```
# Concrete strength over time
days = np.arange(1, 29)
batch_ids = np.repeat(['Batch_A', 'Batch_B', 'Batch_C'], 28)
strengths = np.concatenate([
    np.random.normal(25 + 0.5*days, 2, 28),  # Batch A
    np.random.normal(27 + 0.4*days, 1.5, 28), # Batch B
    np.random.normal(23 + 0.6*days, 2.5, 28) # Batch C
])

qc_df = pd.DataFrame({
    'Day': np.tile(days, 3),
    'Batch': batch_ids,
    'Strength': strengths
})

# Time series with confidence intervals
sns.lineplot(x='Day', y='Strength', hue='Batch',
             data=qc_df, ci=95)
```

Quality Control Output



Key Takeaways

Matplotlib Fundamentals

- **Two interfaces**: MATLAB-style and object-oriented
- **Core plot types**: Line, scatter, histogram, contour
- **Customization**: Colors, styles, layouts
- **Professional output**: Publication-ready figures

Seaborn Advantages

- **Statistical focus**: Built for data exploration
- **Beautiful defaults**: Modern, publication-ready aesthetics
- **DataFrame integration**: Automatic handling of labeled data
- **High-level functions**: Complex plots with simple commands

Resources and References

Documentation

- **Matplotlib**: matplotlib.org
- **Seaborn**: seaborn.pydata.org
- **Gallery**: matplotlib.org/gallery, seaborn.pydata.org/examples
- **Tutorials**: matplotlib.org/tutorials, seaborn.pydata.org/tutorial

Books

- **Python Data Science Handbook** - Jake VanderPlas
- **Matplotlib Plotting Cookbook** - Alexandre Devert
- **Seaborn Documentation** - Official seaborn.pydata.org

Matplotlib & Seaborn Competition

Teams: 4-5 groups of students **Format**: First team to complete each task wins!

Rules

- Work in teams of 4-5 students
- First team to show correct output wins the challenge
- Use Jupyter notebooks for coding
- Ask questions if you need clarification

Prize

The winning team gets a prize!

Challenge 1: Load-Displacement Curve

```
# TASK: Create a professional load-displacement curve
# Requirements:
# 1. Generate displacement data from 0 to 10 mm
# 2. Calculate load using: load = 50 * displacement - 2 * displacement^2
# 3. Add proper labels, title, and grid
# 4. Use matplotlib with seaborn styling
# 5. Mark the yield point at displacement=5mm

# Your code here:
```

First team to show the plot wins Challenge 1!

Challenge 2: Material Strength Comparison

```
# TASK: Compare material strengths using seaborn
# Requirements:
# 1. Create data for 3 materials: Steel, Concrete, Composite
# 2. Generate 100 strength values for each material
# 3. Steel: mean=50, std=5 MPa
# 4. Concrete: mean=30, std=3 MPa
# 5. Composite: mean=40, std=4 MPa
# 6. Use seaborn boxplot to compare distributions
# 7. Add proper labels and title

# Your code here:
```

First team to show the boxplot wins Challenge 2!

Challenge 3: Soil Analysis Regression

```
# TASK: Analyze soil shear strength vs depth
# Requirements:
# 1. Generate depth data from 0 to 20 meters
# 2. Create 3 soil types: Clay, Sand, Silt
# 3. Calculate shear strength: 20 + 2*depth + noise
# 4. Add soil-specific adjustments: Clay +10, Sand -5, Silt +0
# 5. Use seaborn lmplot with hue='soil_type'
# 6. Show regression lines for each soil type

# Your code here:
```

First team to show the regression plot wins Challenge 3!

Challenge 4: Construction Quality Control

```
# TASK: Track concrete strength development over time
# Requirements:
# 1. Create time data from day 1 to 28
# 2. Generate 3 concrete batches with different strength curves
# 3. Batch A: 25 + 0.5*day + noise
# 4. Batch B: 27 + 0.4*day + noise
# 5. Batch C: 23 + 0.6*day + noise
# 6. Use seaborn lineplot with confidence intervals
# 7. Add proper labels and legend

# Your code here:
```

First team to show the time series plot wins Challenge 4!

Challenge 5: Structural Analysis Dashboard

```
# TASK: Create a comprehensive analysis dashboard
# Requirements:
# 1. Create 2x2 subplot layout
# 2. Subplot 1: Load-displacement curve (matplotlib)
# 3. Subplot 2: Material strength distribution (seaborn histogram)
# 4. Subplot 3: Correlation heatmap of structural properties
# 5. Subplot 4: Stress distribution contour plot
# 6. Add overall title and proper spacing
# 7. Use both matplotlib and seaborn in the same figure

# Your code here:
```

First team to show the complete dashboard wins Challenge 5!

Questions?

Thank you for your attention!

Questions and Discussion

Contact: eyuphan.koc@bogazici.edu.tr