Python for Data Science : Week 3 - Tutorial

Visualization of data

Problem 1

Question: Simulating Sales Data

You are working for a retail company, and your task is to simulate sales data for different products. You have a list of products, their base prices, and the number of sales for each product. Your goal is to calculate the total revenue and profit for the company.

```
import numpy as np
import pandas as pd

#========defining=============
# List of products and their base prices
products = ['Product A', 'Product B', 'Product C', 'Product D']
base_prices = [10, 15, 20, 25] #in dollars

# Simulate the number of sales for each product (random integers between 50 and 100)
np.random.seed(42)  # Set a random seed for reproducibility
num_sales = np.random.randint(50, 101, size=len(products))

#=========process1================
# Calculate the total revenue and profit
total_revenue = sum(base_prices[i] * num_sales[i] for i in range(len(products)))
total_cost = sum(5 * num_sales[i] for i in range(len(products)))  # Assuming a constant cost of $5 per unit
total_profit = total_revenue - total_cost

#=========================================

#=======others=========================
# Create a DataFrame to store the sales data
sales_data = pd.DataFrame({
    'Product': products,
    'Base Price': base_prices,
    'Number of Sales': num_sales,
})

# Calculate the revenue and profit for each product
sales_data['Revenue'] = sales_data['Base Price'] * sales_data['Number of Sales']
sales_data['Cost'] = 5 * sales_data['Number of Sales']
sales_data['Profit'] = sales_data['Revenue'] - sales_data['Cost']

print("Sales Data:")
print(sales_data)
print("\nTotal Revenue: $", total_revenue)
print("Total Profit: $", total_profit)
```

```
Sales Data:
     Product  Base Price  Number of Sales  Revenue  Cost  Profit
0  Product A          10               88      880   440     440
1  Product B          15               78     1170   390     780
2  Product C          20               64     1280   320     960
3  Product D          25               92     2300   460    1840

Total Revenue: $ 5630
Total Profit: $ 4020
```

```
np.random.seed?
```

Problem 2

Question: Analyzing Student Grades

Suppose you are given a CSV file containing information about student grades in different subjects. The CSV file has the following columns: "StudentID", "Name", "Subject", and "Grade". Each row represents a student's grade in a specific subject.

Your task is to:

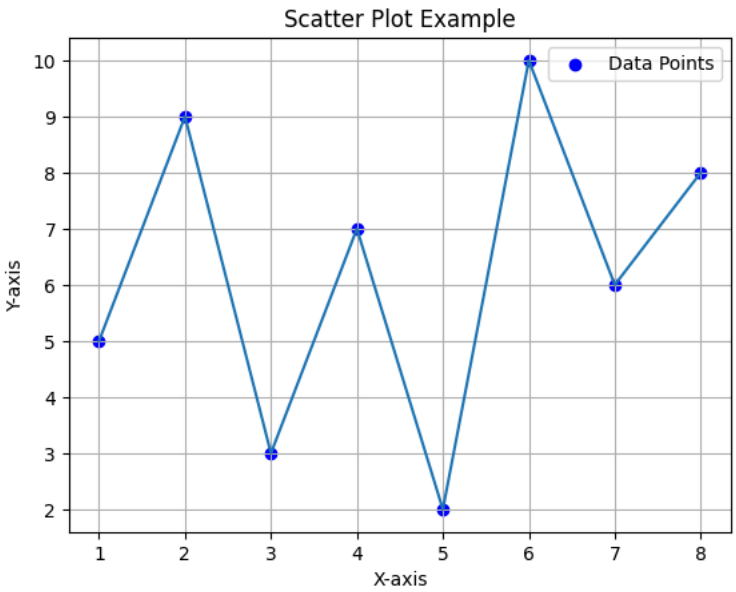
Load the data from the CSV file into a pandas DataFrame. Calculate the average grade for each subject. Find the student with the highest average grade. Determine the number of students who failed (grade less than 60) in each subject. Concatenate the student's name and subject to form a new column "Student_Subject" in the DataFrame. Convert the DataFrame into a NumPy array and perform element-wise arithmetic operations. Use string operations to extract the first name and last name of the student.

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

Plotting

Scatter Plot

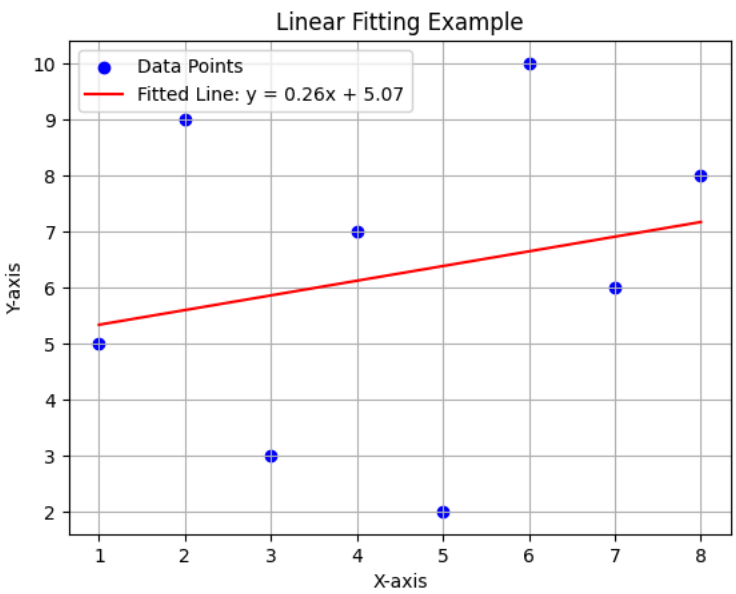
```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5, 6, 7, 8]
y = [5, 9, 3, 7, 2, 10, 6, 8]

# Create a scatter plot
plt.plot(x,y)
plt.scatter(x, y, color='blue', marker='o', label='Data Points')

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot Example')

# Add legend
plt.legend()

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



Linear fitting

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Sample data
x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([5, 9, 3, 7, 2, 10, 6, 8])

# Define the linear function (y = mx + b)
def linear_func(x, m, b):
```

```
    return m * x + b

# Fit the data using curve_fit
params, covariance = curve_fit(linear_func, x, y)

# Get the optimized parameters
m, b = params

# Generate the fitted line
fitted_y = linear_func(x, m, b)

# Create a scatter plot of the data points
plt.scatter(x, y, color='blue', marker='o', label='Data Points')

# Plot the fitted line
plt.plot(x, fitted_y, color='red', label=f'Fitted Line: y = {m:.2f}x + {b:.2f}')

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Linear Fitting Example')

# Add legend
plt.legend()

# Show the plot
plt.grid(True)
plt.show()

# Print the optimized parameters
print(f'Optimized slope (m): {m:.2f}')
print(f'Optimized intercept (b): {b:.2f}')
```

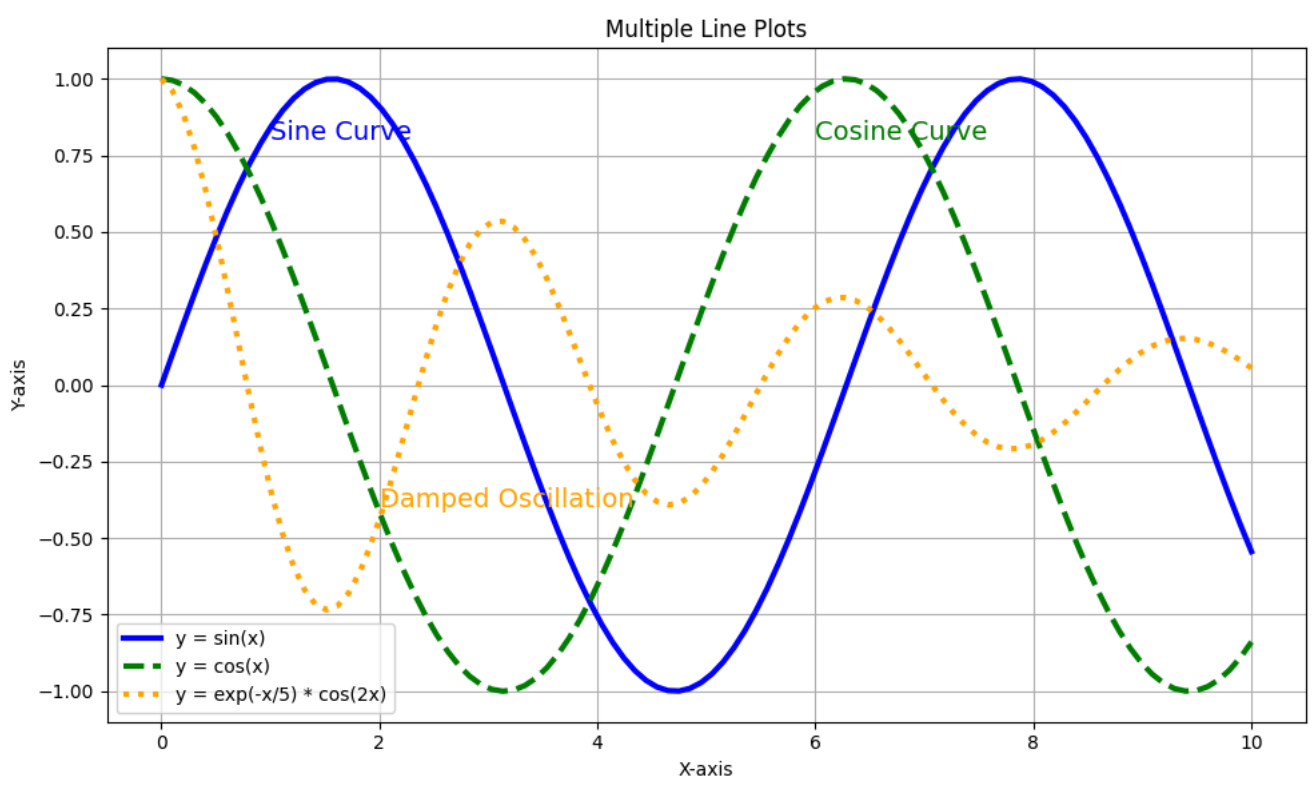



```
Optimized slope (m): 0.26
Optimized intercept (b): 5.07
```

Line Plot

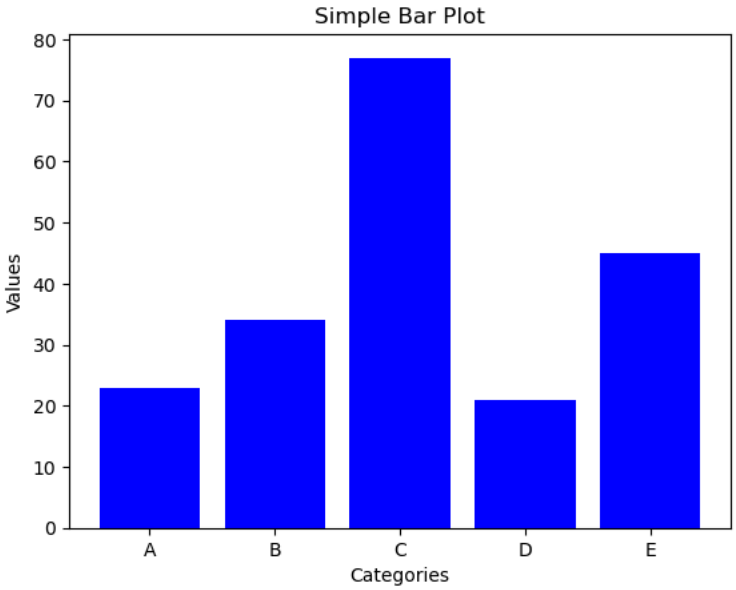
```
import numpy as np
import matplotlib.pyplot as plt

# Generate sample data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
y3 = np.exp(-x/5) * np.cos(2*x)

# Create a new figure and axis
plt.figure(figsize=(10, 6))
ax = plt.subplot()

# Plot the data series
ax.plot(x, y1, label='y = sin(x)', color='blue', linestyle='-', linewidth=3)
ax.plot(x, y2, label='y = cos(x)', color='green', linestyle='--', linewidth=3)
ax.plot(x, y3, label='y = exp(-x/5) * cos(2x)', color='orange', linestyle=':', linewidth=3)
```

```
# Customize plot appearance
ax.set_title('Multiple Line Plots')
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.legend()
ax.grid(True)

# Add annotations
ax.text(1, 0.8, 'Sine Curve', fontsize=14, color='blue')
ax.text(6, 0.8, 'Cosine Curve', fontsize=14, color='green')
ax.text(2, -0.4, 'Damped Oscillation', fontsize=14, color='orange')

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

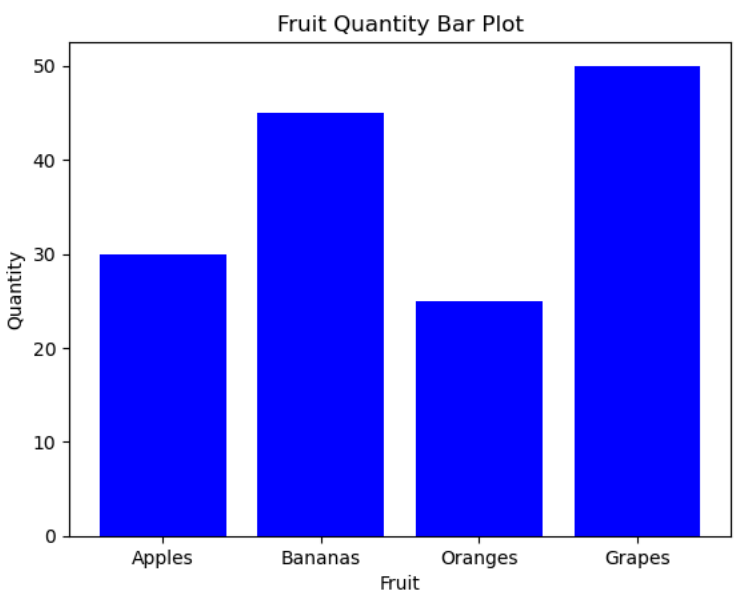


Bar Plot

```
import matplotlib.pyplot as plt

# Sample data
x = [23, 34, 77, 21, 45]
labels = ['A', 'B', 'C', 'D', 'E']

# Create a bar plot
plt.bar(labels, x, color='blue')

# Add labels and title
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Simple Bar Plot')

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

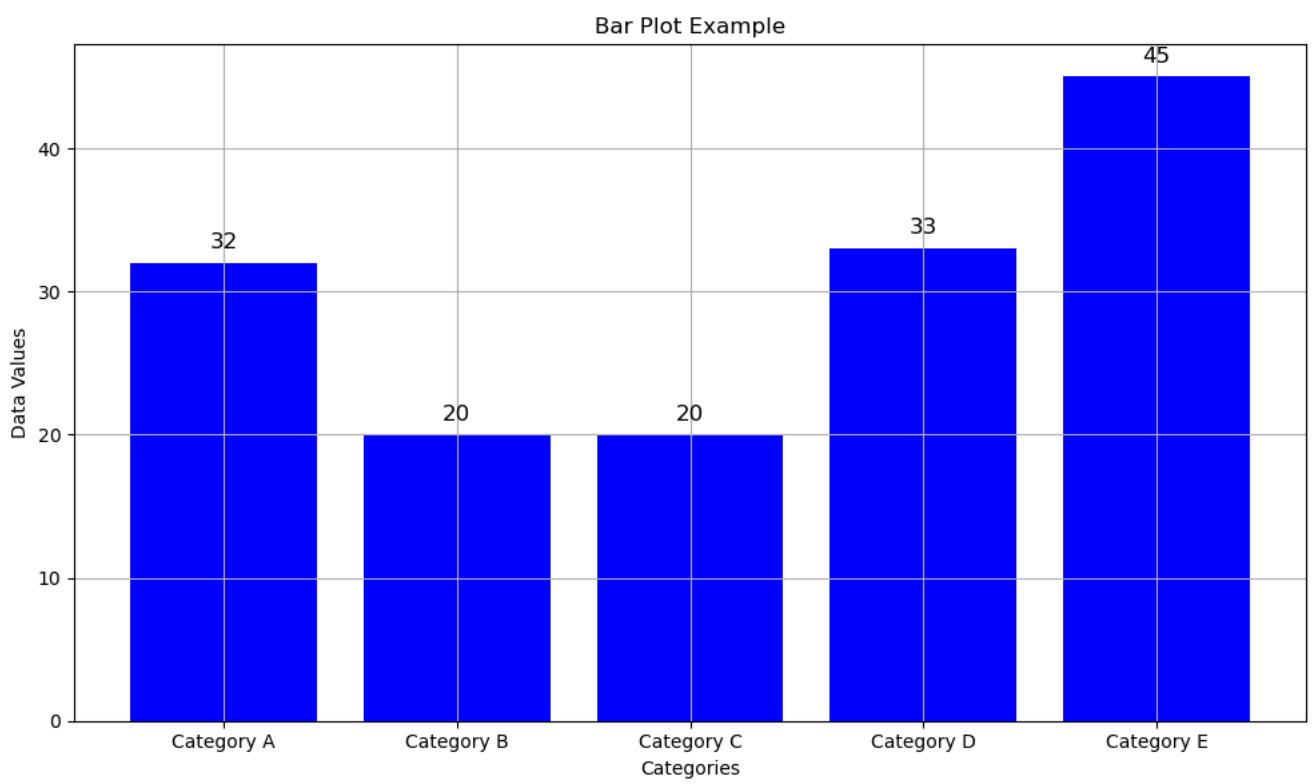



```
# Sample data
categories = ['Apples', 'Bananas', 'Oranges', 'Grapes']
values = [30, 45, 25, 50]

# Create a bar plot
plt.bar(categories, values, color='blue')

# Add labels and title
plt.xlabel('Fruit')
plt.ylabel('Quantity')
plt.title('Fruit Quantity Bar Plot')

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



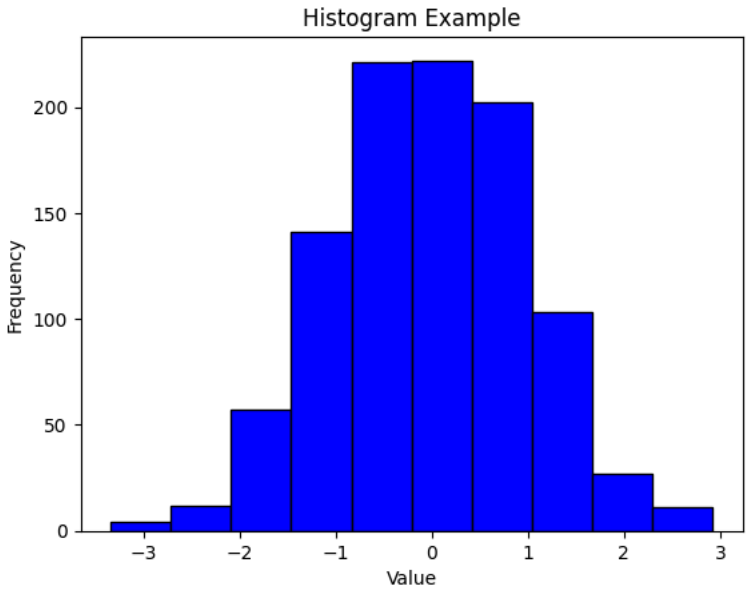

```
# Generate sample data
categories = ['Category A', 'Category B', 'Category C', 'Category D', 'Category E']
data = np.random.randint(10, 50, size=len(categories))

# Create a new figure and axis
plt.figure(figsize=(10, 6))
ax = plt.subplot()

# Create a bar plot
ax.bar(categories, data, color='blue')

# Customize plot appearance
ax.set_title('Bar Plot Example')
ax.set_xlabel('Categories')
ax.set_ylabel('Data Values')
ax.grid(True)
```

```
# Add data values on top of the bars
for i, value in enumerate(data):
    ax.text(i, value + 1, str(value), ha='center', fontsize=12, color='black')

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

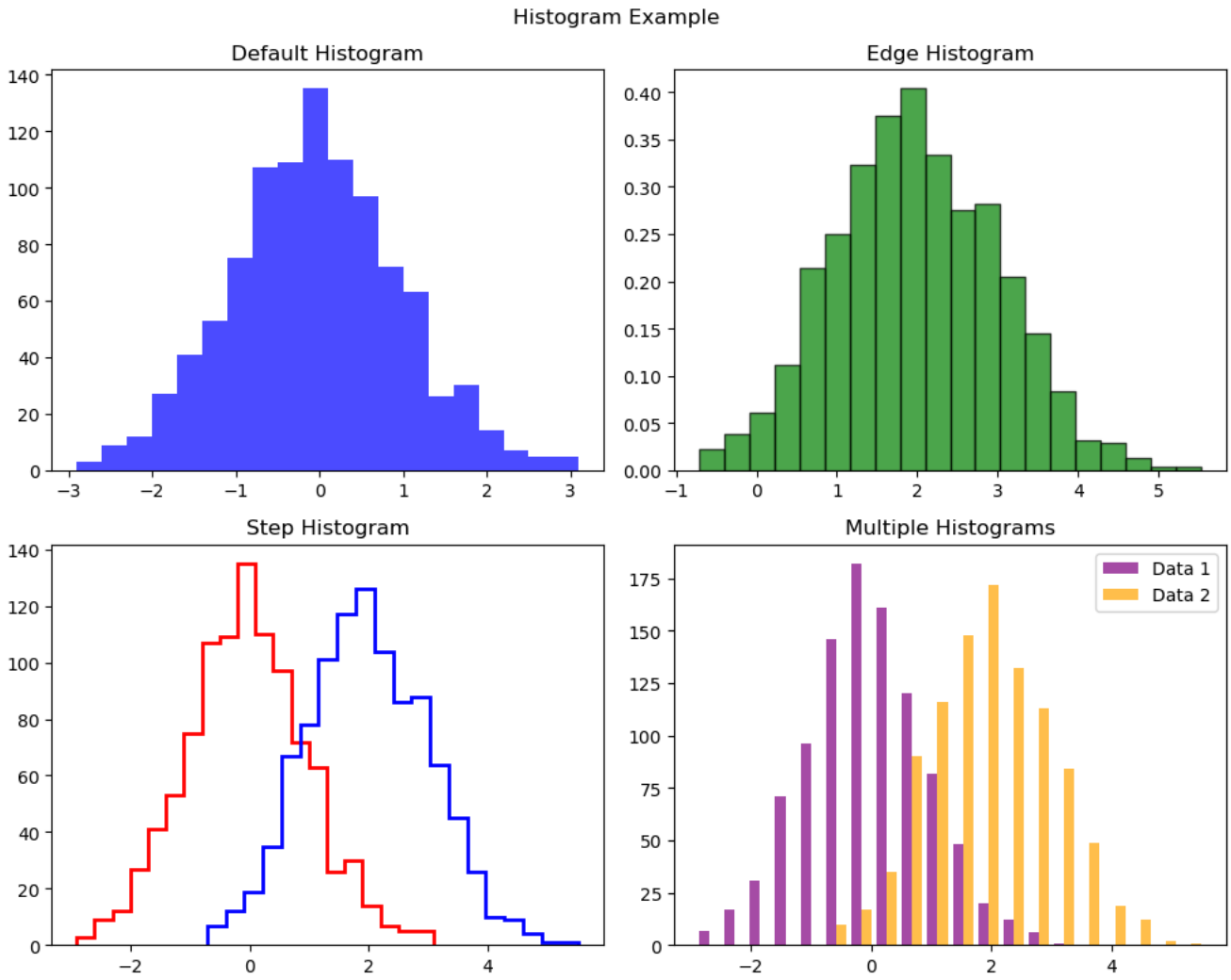


Histogram Plot

```
# Generate random data
data = np.random.randn(1000)  # 1000 random samples from a standard normal distribution

# Create a histogram
plt.hist(data, bins=10, color='blue', edgecolor='black')

# Add labels and title
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram Example')

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



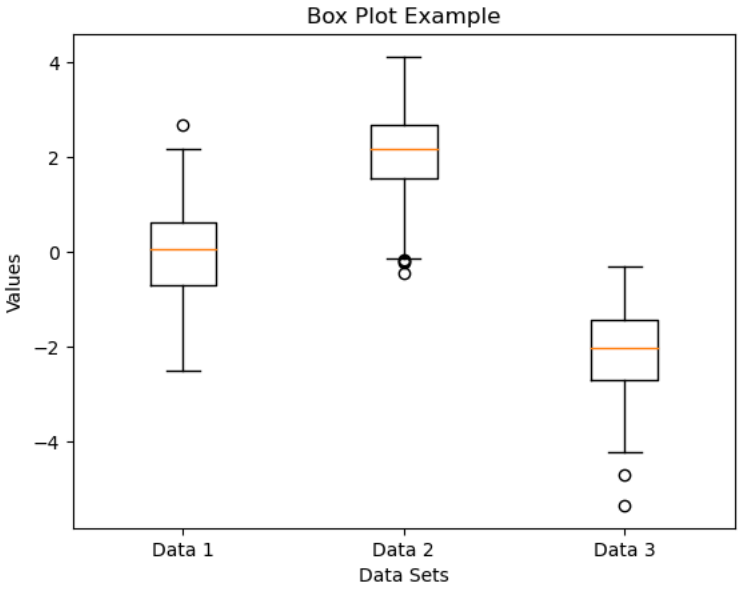

```
import numpy as np
import matplotlib.pyplot as plt

# Generate random data for two distributions
data1 = np.random.normal(0, 1, 1000)
data2 = np.random.normal(2, 1, 1000)

# Create a figure with subplots
fig, axs = plt.subplots(2, 2, figsize=(10, 8))
fig.suptitle('Histogram Example')

# Plot histograms with different customizations
axs[0, 0].hist(data1, bins=20, color='blue', alpha=0.7)
axs[0, 0].set_title('Default Histogram')

axs[0, 1].hist(data2, bins=20, color='green', alpha=0.7, density=True, edgecolor='black')
axs[0, 1].set_title('Edge Histogram')

axs[1, 0].hist(data1, bins=20, color='red', histtype='step', linewidth=2)
axs[1, 0].hist(data2, bins=20, color='blue', histtype='step', linewidth=2)
axs[1, 0].set_title('Step Histogram')

axs[1, 1].hist([data1, data2], bins=20, color=['purple', 'orange'], alpha=0.7, label=['Data 1', 'Data 2'])
axs[1, 1].legend()
axs[1, 1].set_title('Multiple Histograms')

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



Box Plot

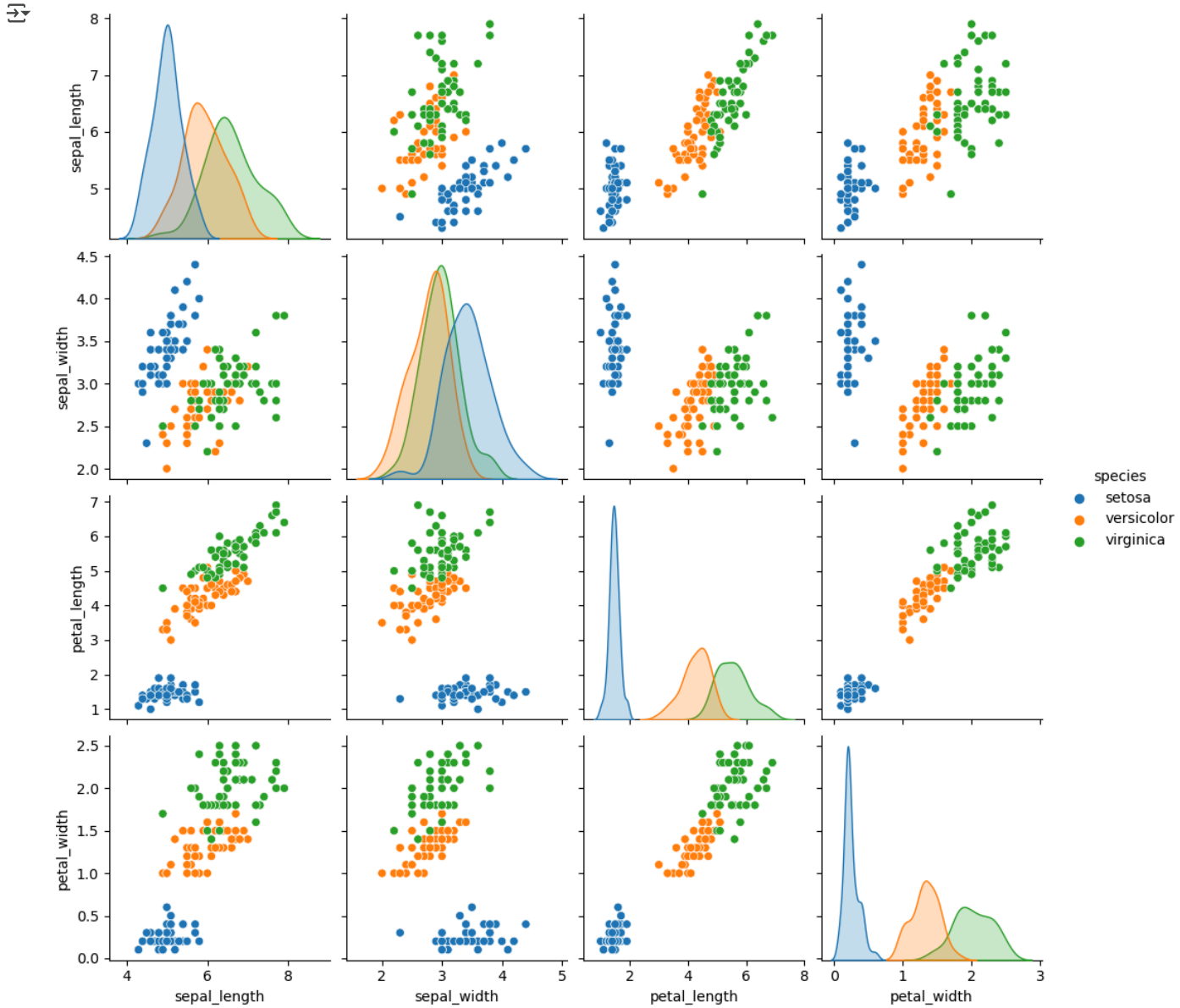
A box plot, also known as a box-and-whisker plot, is a graphical representation of the distribution of a dataset along a number line. It displays summary statistics of the dataset, such as the median, quartiles, and potential outliers. A box plot is particularly useful for identifying the spread and skewness of data.

```
# Generate random data for three distributions
data1 = np.random.normal(0, 1, 100)
data2 = np.random.normal(2, 1, 100)
data3 = np.random.normal(-2, 1, 100)

# Combine the data
data = [data1, data2, data3]

# Create a box plot
plt.boxplot(data, labels=['Data 1', 'Data 2', 'Data 3'])

# Add labels and title
plt.xlabel('Data Sets')
plt.ylabel('Values')
plt.title('Box Plot Example')

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



A pair plot is a way to visualize the relationships between multiple variables in a dataset. It creates scatter plots for all pairs of variables in the dataset, along with histograms along the diagonal to show the distribution of each individual variable. Pair plots are especially useful for exploring correlations and patterns in multivariate data.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load a sample dataset
iris = sns.load_dataset("iris")

# Create a pair plot
sns.pairplot(iris, hue="species")

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




```
sns.load_dataset?
```

```
Object `sns.load_dataset` not found.
```

Loop and algorithms

if-else, elif

```
# User input
age = int(input("Enter your age: "))

# Check if the user is eligible to vote
if age >= 18:
    print("You are eligible to vote.")
else:
    print("You are not eligible to vote yet.")
```

```
Enter your age: 10
You are not eligible to vote yet.
```

```
# User input
number = float(input("Enter a number: "))

# Check if the number is positive, negative, or zer
```

```
if number > 0:
    print("The number is positive.")
elif number < 0:
    print("The number is negative.")
else:
```

```
Enter a number: 12
The number is positive.
```

for loop

```
# Using a for-loop to print numbers
for num in range(1, 6): #start = what you give , end = n-1
    print(num)
```

```
1
2
3
4
```