# Task 1: NumPy Basics

Write a Python script that creates a NumPy array with random integers and performs basic mathematical operations like addition, subtraction, multiplication, and division on it.

import numpy as np

# Define the size of the array and the range for random integers

array\_size = (4, 4) # You can change the dimensions as needed

min\_value = 1

max\_value = 10

# Create a random integer array

random\_array = np.random.randint(min\_value, max\_value + 1, size=array\_size)

print("Random Array:")

print(random\_array)

# Perform basic mathematical operations

scalar = 2 # You can change the scalar value as needed

# Addition

addition\_result = random\_array + scalar

print("\nAddition Result:")

print(addition\_result)

# Subtraction

subtraction\_result = random\_array - scalar

print("\nSubtraction Result:")

print(subtraction\_result)

# Multiplication

multiplication\_result = random\_array \* scalar

print("\nMultiplication Result:")

print(multiplication\_result)

# Division

division\_result = random\_array / scalar

print("\nDivision Result:")

print(division\_result)

# Task 2: Pandas Data Analysis

Using Pandas, load a dataset of your choice (you can find datasets online or use your own). Clean the data and perform basic data analysis tasks like calculating mean, median, and mode for specific columns.

import pandas as pd

# Load the Iris dataset (this is a sample dataset available in Pandas)

iris\_df = pd.read\_csv("https://raw.githubusercontent.com/pandas-dev/pandas/master/pandas/tests/data/iris.csv")

# Display the first few rows of the dataset to inspect its structure

print("First few rows of the Iris dataset:")

print(iris\_df.head())

# Clean the data (optional but useful)

# For this example, we assume that the dataset is already clean

# Calculate mean, median, and mode for specific columns

# Let's calculate these statistics for the 'sepal\_length' column as an example

sepal\_length\_mean = iris\_df['sepal\_length'].mean()

sepal\_length\_median = iris\_df['sepal\_length'].median()

sepal\_length\_mode = iris\_df['sepal\_length'].mode().values[0]

print("\nStatistics for 'sepal\_length' column:")

print(f"Mean: {sepal\_length\_mean}")

print(f"Median: {sepal\_length\_median}")

print(f"Mode: {sepal\_length\_mode}")

# Task 3: Data Visualization

Using Matplotlib, create a bar chart or histogram to visualize the distribution of a dataset of your choice. Label the axes and add a title to the chart.

import numpy as np

import matplotlib.pyplot as plt

# Generate a random dataset (replace this with your own data)

np.random.seed(0)

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

# Create a histogram

plt.hist(data, bins=20, color='skyblue', edgecolor='black')

# Label the axes and add a title

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.title('Histogram of Random Data')

# Show the plot

plt.show()

# Task 4: NumPy Array Manipulation

Create two NumPy arrays and perform operations to concatenate them vertically and horizontally.

import numpy as np

# Create two sample arrays

array1 = np.array([[1, 2], [3, 4]])

array2 = np.array([[5, 6]])

# Vertical concatenation using numpy.concatenate

vertical\_concatenation = np.concatenate((array1, array2), axis=0)

# Horizontal concatenation using numpy.concatenate

# Note: For horizontal concatenation, the arrays must have the same number of rows.

array3 = np.array([[7], [8]])

horizontal\_concatenation = np.concatenate((array1, array3), axis=1)

# Alternatively, you can use numpy.vstack and numpy.hstack for stacking

vertical\_stack = np.vstack((array1, array2))

horizontal\_stack = np.hstack((array1, array3))

print("Original Array 1:")

print(array1)

print("\nOriginal Array 2:")

print(array2)

print("\nVertical Concatenation:")

print(vertical\_concatenation)

print("\nHorizontal Concatenation:")

print(horizontal\_concatenation)

# Using vstack and hstack

print("\nUsing vstack:")

print(vertical\_stack)

print("\nUsing hstack:")

print(horizontal\_stack)

# Task 5: Pandas Data Filtering

Given a dataset (you can choose one or use your own), use Pandas to filter and extract rows that meet specific criteria (e.g., filtering data for a specific date range).

import pandas as pd

# Sample dataset with date values

data = {'Date': ['2023-09-01', '2023-09-02', '2023-09-03', '2023-09-04', '2023-09-05'],

'Value': [10, 15, 8, 12, 18]}

# Create a DataFrame

df = pd.DataFrame(data)

# Convert the 'Date' column to datetime format

df['Date'] = pd.to\_datetime(df['Date'])

# Define the date range for filtering

start\_date = '2023-09-02'

end\_date = '2023-09-04'

# Filter and extract rows within the date range

filtered\_df = df[(df['Date'] >= start\_date) & (df['Date'] <= end\_date)]

# Display the filtered DataFrame

print("Original DataFrame:")

print(df)

print("\nFiltered DataFrame:")

print(filtered\_df)

# Task 6: Matplotlib Customization

Create a line plot or scatter plot using Matplotlib and customize it by adding a legend, different marker styles, and changing the line colors.

import matplotlib.pyplot as plt

# Sample data

x\_data = [1, 2, 3, 4, 5]

y1\_data = [2, 4, 1, 3, 5]

y2\_data = [5, 3, 2, 4, 1]

# Create a scatter plot

plt.scatter(x\_data, y1\_data, label='Series 1', color='blue', marker='o')

plt.scatter(x\_data, y2\_data, label='Series 2', color='red', marker='x')

# Customize the plot

plt.title('Scatter Plot with Legend')

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

# Add a legend

plt.legend(loc='upper left')

# Show the plot

plt.grid(True)

plt.show()

# Task 7: NumPy Statistical Analysis

Use NumPy to calculate descriptive statistics like variance, standard deviation, and correlation coefficients for a dataset of your choice (you can find datasets online or use your own).

import numpy as np

# Sample dataset 1

dataset1 = np.array([10, 15, 20, 25, 30])

# Sample dataset 2

dataset2 = np.array([5, 8, 12, 16, 20])

# Calculate variance and standard deviation for dataset1

variance1 = np.var(dataset1)

std\_deviation1 = np.std(dataset1)

# Calculate variance and standard deviation for dataset2

variance2 = np.var(dataset2)

std\_deviation2 = np.std(dataset2)

# Calculate the correlation coefficient between dataset1 and dataset2

correlation\_coefficient = np.corrcoef(dataset1, dataset2)[0, 1]

# Display the results

print("Descriptive Statistics for Dataset 1:")

print(f"Variance: {variance1}")

print(f"Standard Deviation: {std\_deviation1}")

print("\nDescriptive Statistics for Dataset 2:")

print(f"Variance: {variance2}")

print(f"Standard Deviation: {std\_deviation2}")

print("\nCorrelation Coefficient between Dataset 1 and Dataset 2:")

print(f"Correlation Coefficient: {correlation\_coefficient}")

# Task 8: Pandas Data Grouping

Given a dataset (you can choose one or use your own), use Pandas to group data by a specific column and calculate summary statistics (e.g., mean, median) for each group.

import pandas as pd

# Sample dataset (you can replace this with your own dataset)

data = {

'Student\_ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'Subject': ['Math', 'Science', 'Math', 'Science', 'Math', 'Science', 'Math', 'Science', 'Math', 'Science'],

'Score': [85, 92, 78, 88, 90, 89, 76, 85, 80, 87]

}

# Create a DataFrame

df = pd.DataFrame(data)

# Group the data by the 'Subject' column

grouped = df.groupby('Subject')

# Calculate summary statistics for each group

summary\_stats = grouped.agg({'Score': ['mean', 'median']})

# Display the summary statistics

print(summary\_stats)

# Task 9: Matplotlib Subplots

Create a Matplotlib figure with multiple subplots to display different aspects of a dataset or related datasets (you can choose the dataset and the type of plots).

import matplotlib.pyplot as plt

import seaborn as sns # Import Seaborn for data visualization

import pandas as pd

# Load the Iris dataset (you can replace this with your own dataset)

iris = sns.load\_dataset("iris")

# Create a Matplotlib figure with subplots

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Subplot 1: Scatter plot of sepal length vs. sepal width

sns.scatterplot(data=iris, x="sepal\_length", y="sepal\_width", hue="species", ax=axes[0])

axes[0].set\_title("Scatter Plot: Sepal Length vs. Sepal Width")

# Subplot 2: Histogram of petal length

sns.histplot(data=iris, x="petal\_length", bins=20, kde=True, ax=axes[1])

axes[1].set\_title("Histogram: Petal Length Distribution")

# Subplot 3: Box plot of petal width

sns.boxplot(data=iris, x="species", y="petal\_width", ax=axes[2])

axes[2].set\_title("Box Plot: Petal Width by Species")

# Adjust spacing between subplots

plt.tight\_layout()

# Show the figure

plt.show()

# Task 10: Pandas Data Visualization

Choose a dataset of your choice and create various visualizations (e.g., bar charts, pie charts, scatter plots) using Pandas to represent different aspects of the data, such as trends or distributions.

import pandas as pd

import matplotlib.pyplot as plt

# Sample dataset

data = {

'Category': ['Food', 'Rent', 'Utilities', 'Transportation', 'Entertainment'],

'Expense': [500, 1200, 300, 200, 100]

}

# Create a DataFrame

expenses\_df = pd.DataFrame(data)

# Visualization 1: Bar chart for expense distribution

plt.figure(figsize=(8, 5))

plt.bar(expenses\_df['Category'], expenses\_df['Expense'], color='skyblue')

plt.xlabel('Expense Category')

plt.ylabel('Expense Amount ($)')

plt.title('Monthly Expense Distribution')

plt.xticks(rotation=45)

plt.tight\_layout()

# Visualization 2: Pie chart for expense proportions

plt.figure(figsize=(6, 6))

plt.pie(expenses\_df['Expense'], labels=expenses\_df['Category'], autopct='%1.1f%%', colors=['lightcoral', 'lightblue', 'lightgreen', 'lightsalmon', 'lightpink'])

plt.title('Expense Proportions')

plt.tight\_layout()

# Visualization 3: Scatter plot for expense trends

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May']

expenses\_df['Month'] = months

plt.figure(figsize=(8, 5))

plt.scatter(expenses\_df['Month'], expenses\_df['Expense'], color='purple', marker='o', s=100)

plt.xlabel('Month')

plt.ylabel('Expense Amount ($)')

plt.title('Expense Trends Over Months')

plt.grid(True)

# Show the plots

plt.show()





