Exercise 1: Introduction to NumPy and Arrays

# 1. Creating NumPy Arrays with Filtering

import numpy as np

# Load the housing data from the CSV file into a NumPy array

housing\_data = np.loadtxt('housing.csv', delimiter=',')

# Use advanced indexing to extract only the rows where the 'price' is above a certain threshold (e.g., $1 million)

filtered\_housing\_data = housing\_data[housing\_data[:, 6] > 1000000]

# Calculate and print the mean price of the filtered houses

mean\_price = np.mean(filtered\_housing\_data[:, 6])

print('Mean price of filtered houses:', mean\_price)

# 2. Statistical Analysis with NumPy

# Load the housing data into a NumPy array

housing\_data = np.loadtxt('housing.csv', delimiter=',')

# Calculate and print the 25th, 50th (median), and 75th percentiles of the 'sqft\_living' column using NumPy functions

percentiles = np.percentile(housing\_data[:, 5], [25, 50, 75])

print('Percentiles of sqft\_living:', percentiles)

# Create a boolean mask to filter and print the rows where 'sqft\_living' is above the 75th percentile

sqft\_living\_mask = housing\_data[:, 5] > percentiles[2]

filtered\_housing\_data = housing\_data[sqft\_living\_mask]

print('Filtered housing data where sqft\_living is above the 75th percentile:', filtered\_housing\_data)

Exercise 2: Attributes of NumPy Arrays

# 1. Advanced Array Attributes and Methods

# Load the housing data into a NumPy array

housing\_data = np.loadtxt('housing.csv', delimiter=',')

# Use advanced indexing to extract rows where 'bedrooms' is greater than 5

filtered\_housing\_data = housing\_data[housing\_data[:, 4] > 5]

# Calculate and print the mean 'price' and 'sqft\_living' for the selected rows

mean\_price = np.mean(filtered\_housing\_data[:, 6])

mean\_sqft\_living = np.mean(filtered\_housing\_data[:, 5])

print('Mean price for houses with more than 5 bedrooms:', mean\_price)

print('Mean sqft\_living for houses with more than 5 bedrooms:', mean\_sqft\_living)

# 2. Advanced Reshaping and Transposing

# Create a 2D NumPy array with random values of shape (5, 4)

random\_array = np.random.rand(5, 4)

# Reshape it into a 1D array (flattened)

flattened\_array = random\_array.flatten()

# Transpose the original 2D array

transposed\_array = random\_array.T

# Print the reshaped and transposed arrays

print('Reshaped array:', flattened\_array)

print('Transposed array:', transposed\_array)

Exercise 3: Array Indexing and Slicing

# 1. Advanced Indexing with Multiple Conditions

# Load the housing data into a NumPy array

housing\_data = np.loadtxt('housing.csv', delimiter=',')

# Use advanced indexing to extract rows where 'bedrooms' is greater than 4 and 'bathrooms' is greater than 3

filtered\_housing\_data = housing\_data[(housing\_data[:, 4] > 4) & (housing\_data[:, 3] > 3)]

# Calculate and print the mean 'price' for the selected rows

mean\_price = np.mean(filtered\_housing\_data[:, 6])

print('Mean price for houses with more than 4 bedrooms and 3 bathrooms:', mean\_price)

# 2. Advanced Slicing with Conditions

# Create a 2D NumPy array with random values

random\_array = np.random.rand(5, 5)

# Use slicing and advanced indexing to extract rows where the sum of values in each row is greater than a certain threshold

threshold = 10

filtered\_random\_array = random\_array[np.sum(random\_array, axis=1) > threshold]

# Print the selected rows and their sums

print('Filtered rows’, filtered\_random\_array)

Exercise 4: Array Operations and Functions

#1. Advanced Element-Wise Operations:

# Create two 2D NumPy arrays of the same shape (e.g., (3, 3)) with random values

array\_1 = np.random.rand(3, 3)

array\_2 = np.random.rand(3, 3)

# Perform element-wise matrix multiplication between the two arrays

matrix\_multiplication = array\_1 \* array\_2

# Print the result

print('Element-wise matrix multiplication:', matrix\_multiplication)

2. Advanced Array Functions and Aggregation

import numpy as np

# Load the housing data into a NumPy array

housing\_data = np.loadtxt('housing.csv', delimiter=',')

# Use NumPy functions like np.unique() to find unique values in the 'zipcode' column

unique\_zipcodes = np.unique(housing\_data[:, 1])

# Calculate and print the mean 'price' for each unique 'zipcode'

mean\_price\_per\_zipcode = []

for zipcode in unique\_zipcodes:

zipcode\_mask = housing\_data[:, 1] == zipcode

zipcode\_data = housing\_data[zipcode\_mask]

mean\_price\_per\_zipcode.append(np.mean(zipcode\_data[:, 6]))

# Print the results

print('Mean price for each unique zipcode:', mean\_price\_per\_zipcode)

## Exercise 5: Data Science Operations

#1. Advanced Stacking and Broadcasting

import numpy as np

import matplotlib.pyplot as plt

# Create a 2D NumPy array representing a grayscale image (pixel values between 0 and 255)

image = np.random.randint(0, 256, size=(256, 256))

# Create a 1D array representing a filter (e.g., a convolution kernel)

filter = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])

# Use broadcasting to apply the filter to the image, resulting in a new filtered image

filtered\_image = np.conv2d(image, filter, mode='same')

# EXTRA

# Visualize the original image and the filtered image side by side

plt.subplot(1, 2, 1)

plt.imshow(image, cmap='gray')

plt.title('Original image')

plt.subplot(1, 2, 2)

plt.imshow(filtered\_image, cmap='gray')

plt.title('Filtered image')

plt.show()

#2. Advanced Vectorization with Functions

# Create a large NumPy array of random values

random\_array = np.random.rand(1000000)

# Define a custom NumPy ufunc (universal function) that computes a custom mathematical operation on each element of the array

def my\_custom\_ufunc(x):

return x \*\* 2 + 1

# Apply the custom ufunc to the array and print the result

result = np.frompyfunc(my\_custom\_ufunc, 1, 1)(random\_array)

print(result)