Learning Objectives

- Demonstrate problem solving
- Use functions, list comprehension, loops
- Comparing with numpy

Problem 1: Vector Dot Product Computation

Vector dot product computation is an immensely useful operation in data science. It allows obtaining one scaler by operating on two vectors. Complete the below `vector_dot_product(a, b)` function without using numpy or any other Python package.

```
def vector_dot_product(a, b):
    """
    The function computes and returns the dot product between two vectors.

    a: list of values denoting elements of a vector
    b: list of values denoting elements of a vector

    Returns:
        y: floating-point number representing the dot product
           or, None if the lengths of the vectors are unequal
    """
    # use try catch to return None if the length of lists is unequal
    # write your code
    y = 0
    return y
# Demonstrate Usage
a = [1, 2, 3]
b = [4, 5, 6]
result = vector_dot_product(a, b)
print(result)
# Example with unequal vector lengths
c = [2, 4, 6, 8]
d = [1, 3, 5]
result = vector_dot_product(c, d)
print(result)
```

Problem 2: Matrix-Vector Multiplication

Write a Python function `matrix_vector_multiply(matrix, vector)` that takes a 2D list `matrix` (representing a matrix) and a 1D list `vector` as input. The function should perform the following tasks:

1. Check if the number of columns in the `matrix` is equal to the length of the `vector`. If not, raise a `ValueError` with an appropriate error message.
2. Implement the matrix-vector multiplication operation from first principles, using nested loops.

Complete the below code.

```python
def matrix_vector_multiply(matrix, vector):
    """
    Performs matrix-vector multiplication from first principles.

    Args:
        matrix (list): A 2D list representing the matrix.
        vector (list): A 1D list representing the vector.
    Returns:
        list: The resulting vector after matrix-vector multiplication.
    Raises:
        ValueError: count of cols in the matrix is not equal to len of the vector.
    """
    # Check if the number of columns in the matrix is equal to the length of the
vector
    num_cols = len(matrix[0])
    if num_cols != len(vector):
        raise ValueError("Alert: Col Count in matrix not matching length of vector")

    # Initialize the resulting vector with zeros
    result = [0] * len(matrix)

    # Perform matrix-vector multiplication and update result
    # write the code, you may reuse the vector dot product here

    return result


# Example usage
# Example 1
matrix = [[1, 2], [3, 4]]
vector = [5, 6]
result = matrix_vector_multiply(matrix, vector)
print(result)

# Example 2
matrix = [[1, 2, 3], [4, 5, 6]]
vector = [7, 8]
try:
    result = matrix_vector_multiply(matrix, vector)
except ValueError as e:
    print(e)  # Output: Alert: ...
```

Problem 3: Matrix-Matrix Multiplication
Complete the below code.

```python
def matrix_multiply(matrix1, matrix2):
```

```
    """
    Performs matrix-matrix multiplication from first principles.
    Args:
        matrix1 (list): A 2D list representing the first matrix.
        matrix2 (list): A 2D list representing the second matrix.
    Returns:
        list: The resulting matrix after matrix-matrix multiplication.
    Raises:
        ValueError: If the number of columns in the first matrix is not equal to
the number of rows in the second matrix.
    """
    # Check if the dimensions are compatible for matrix multiplication
    # write the code

    # Initialize the resulting matrix with zeros
    # write the code

    # Perform matrix-matrix multiplication
    # write the code, you may reuse the matrix-vector here
    return result

# Example usage
# Example 1
matrix1 = [[1, 2], [3, 4]]
matrix2 = [[5, 6], [7, 8]]
result = matrix_multiply(matrix1, matrix2)
print(result)  # Output: [[19, 22], [43, 50]]

# Example 2
matrix1 = [[1, 2, 3], [4, 5, 6]]
matrix2 = [[7, 8], [9, 10], [11, 12]]
result = matrix_multiply(matrix1, matrix2)
print(result)  # Output: [[58, 64], [139, 154]]

# Example 3
matrix1 = [[1, 2], [3, 4]]
matrix2 = [[5, 6, 7], [8, 9, 10]]
try:
    result = matrix_multiply(matrix1, matrix2)
except ValueError as e:
    print(e)  # Output: Alert ...
```

Problem 4: Computation time comparison

Numpy is a powerful Python package to handle numeric data. It is specialized to do matrix-vector multiplications efficiently. You will now compare the computation time for the above task with and without using numpy. Complete the below code.

```
import time
import numpy as np

# Generate a random matrix and vector
matrix_size = 1000
vector_size = 1000
# create a matrix with elements chosen randomly from 1-9
matrix = [[np.random.randint(1, 10) for _ in range(vector_size)] for _ in
range(matrix_size)]
# create a vector with elements chosen randomly from 1-9
# write code

# matrix vector multiplication using List implementation
start_time = time.time()
result_list = matrix_vector_product(matrix, vector)
end_time = time.time()
list_time = end_time - start_time
print(f"List implementation time: {list_time:.6f} seconds")

# NumPy implementation
start_time = time.time()
matrix_np = np.array(matrix)
vector_np = np.array(vector)
# write the one line to multiply matrix and vector
end_time = time.time()
numpy_time = end_time - start_time
print(f"NumPy implementation time: {numpy_time:.6f} seconds")

# Check if the results are the same
assert np.array_equal(result_list, result_np)
print("Results are the same")
```

Problem 5: Law of Large Numbers Demonstration

Write a Python program that simulates the Law of Large Numbers using NumPy. The Law of Large Numbers states that **as the number of trials in a random experiment increases, the sample mean approaches the theoretical mean or expected value**. Use print statements to show how the sample mean changes as the number of trials increases. Choose an experiment of your choice, you can use the random package in numpy to generate a sample population. Hint: use Bernoulli for a coin toss experiment.
