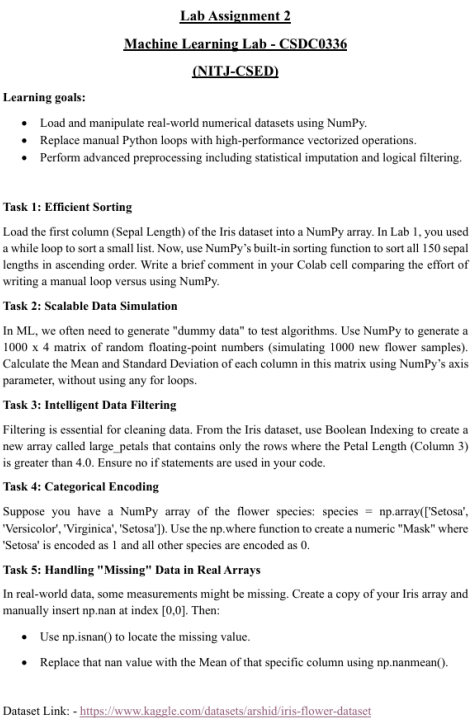
Lab 02 || 23103163



Lab Assignment 2

Machine Learning Lab - CSDC0336

(NITJ-CSED)

Learning goals:

- Load and manipulate real-world numerical datasets using NumPy.
- Replace manual Python loops with high-performance vectorized operations.
- Perform advanced preprocessing including statistical imputation and logical filtering.

Task 1: Efficient Sorting

Load the first column (Sepal Length) of the Iris dataset into a NumPy array. In Lab 1, you used a while loop to sort a small list. Now, use NumPy's built-in sorting function to sort all 150 sepal lengths in ascending order. Write a brief comment in your Colab cell comparing the effort of writing a manual loop versus using NumPy.

Task 2: Scalable Data Simulation

In ML, we often need to generate "dummy data" to test algorithms. Use NumPy to generate a 1000 x 4 matrix of random floating-point numbers (simulating 1000 new flower samples). Calculate the Mean and Standard Deviation of each column in this matrix using NumPy's axis parameter, without using any for loops.

Task 3: Intelligent Data Filtering

Filtering is essential for cleaning data. From the Iris dataset, use Boolean Indexing to create a new array called large_petals that contains only the rows where the Petal Length (Column 3) is greater than 4.0. Ensure no if statements are used in your code.

Task 4: Categorical Encoding

Suppose you have a NumPy array of the flower species: species = np.array(['Setosa', 'Versicolor', 'Virginica', 'Setosa']). Use the np.where function to create a numeric "Mask" where 'Setosa' is encoded as 1 and all other species are encoded as 0.

Task 5: Handling "Missing" Data in Real Arrays

In real-world data, some measurements might be missing. Create a copy of your Iris array and manually insert np.nan at index [0,0]. Then:

- Use np.isnan() to locate the missing value.
- Replace that nan value with the Mean of that specific column using np.nanmean().

Dataset Link: - https://www.kaggle.com/datasets/arshid/iris-flower-dataset

```
numbers = [5, 2, 8, 1, 9, 4]
print(f"Original list: {numbers}")

# Bubble sort using while loop
length = len(numbers)
has_swapped = True

while has_swapped:
    has_swapped = False
    index = 0
    while index < length - 1:
        if numbers[index] > numbers[index + 1]:
            # Swap elements
            numbers[index], numbers[index + 1] = numbers[index + 1], numbers[index]
            ha
```

```
Original list: [5, 2, 8, 1, 9, 4]
Sorted list using while loop: [1, 2, 4, 5, 8, 9]
```

Sorting 'sepal_length_array' using NumPy's built-in sorting function

```
sorted_sepal_length = np.sort(sepal_length_array)

print("First 10 values of the sorted 'sepal_length' NumPy array:")
print(sorted_sepal_length[:150])
print(f"Shape of the sorted array: {sorted_sepal_length.shape}")
```

```
First 10 values of the sorted 'sepal_length' NumPy array:
[4.3 4.4 4.4 4.4 4.5 4.6 4.6 4.6 4.6 4.7 4.7 4.8 4.8 4.8 4.8 4.8 4.9 4.9
 4.9 4.9 4.9 4.9 5.  5.  5.  5.  5.  5.  5.  5. ]
Shape of the sorted array: (150,)
```

```
# Generate 1000x4 matrix of random floats
data = np.random.rand(1000, 4)

# Mean of each column
column_means = np.mean(data, axis=0)

# Standard deviation of each column
column_stds = np.std(data, axis=0)

print("Means:", column_means)
print("Standard Deviations:", column_stds)
```

```
Means: [0.49615572 0.50823375 0.50109201 0.49592796]
Standard Deviations: [0.28321529 0.2889877  0.29355055 0.28233879]
```

```
import numpy as np
from sklearn.datasets import load_iris

# Load Iris dataset
iris = load_iris()

# Extract first column (Sepal Length)
sepal_lengths = iris.data[:, 0]
```

```
# Sort using NumPy built-in
function
sorted_sepal_lengths =
np.sort(sepal_lengths)

print("Sorted Sepal Lengths:")
print(sorted_sepal_lengths)
```

```
Sorted Sepal Lengths:
[4.3 4.4 4.4 4.4 4.5 4.6 4.6 4.6 4.6 4.7 4.7 4.8 4.8 4.8 4.8 4.8 4.9 4.9
 4.9 4.9 4.9 4.9 5.  5.  5.  5.  5.  5.  5.  5.  5.  5.  5.  5.1 5.1 5.1 5.1
 5.1 5.1 5.1 5.1 5.1 5.2 5.2 5.2 5.2 5.3 5.4 5.4 5.4 5.4 5.4 5.4 5.5 5.5
 5.5 5.5 5.5 5.5 5.5 5.6 5.6 5.6 5.6 5.6 5.6 5.7 5.7 5.7 5.7 5.7 5.7 5.7
 5.7 5.8 5.8 5.8 5.8 5.8 5.8 5.8 5.9 5.9 5.9 6.  6.  6.  6.  6.  6.  6.1
 6.1 6.1 6.1 6.1 6.2 6.2 6.2 6.2 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3
 6.4 6.4 6.4 6.4 6.4 6.4 6.4 6.5 6.5 6.5 6.5 6.5 6.6 6.6 6.7 6.7 6.7 6.7
 6.7 6.7 6.7 6.8 6.8 6.8 6.9 6.9 6.9 6.9 7.  7.1 7.2 7.2 7.2 7.3 7.4
 7.6 7.7 7.7 7.7 7.7 7.9]
```

Comment :

NumPy's built-in sorting is faster and more efficient than manual loops because it uses optimized C-based implementations and avoids Python-level iteration.

```
import numpy as np
from sklearn.datasets import
load_iris

# Load Iris dataset
iris = load_iris()

# Create Boolean mask for Petal
Length > 4.0 (Column index 2)
mask = iris.data[:, 2] > 4.0

# Filter rows using Boolean
indexing
large_petals = iris.data[mask]

print("Filtered data (Petal Length
> 4.0):")
print(large_petals)
```

```
Filtered data (Petal Length > 4.0):
[[7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [6.5 2.8 4.6 1.5]
 [5.7 2.8 4.5 1.3]
 [6.3 3.3 4.7 1.6]
 [6.6 2.9 4.6 1.3]
 [5.9 3.  4.2 1.5]
 [6.1 2.9 4.7 1.4]
 [6.7 3.1 4.4 1.4]
 [5.6 3.  4.5 1.5]
 [5.8 2.7 4.1 1. ]
 [6.2 2.2 4.5 1.5]
 [5.9 3.2 4.8 1.8]
 [6.3 2.5 4.9 1.5]
 [6.1 2.8 4.7 1.2]
 [6.4 2.9 4.3 1.3]
 [6.6 3.  4.4 1.4]
 [6.8 2.8 4.8 1.4]
 [6.7 3.  5.  1.7]
 [6.  2.9 4.5 1.5]
 [6.  2.7 5.1 1.6]
 [5.4 3.  4.5 1.5]
 [6.  3.4 4.5 1.6]
```

```
# Given species array
species = np.array(['Setosa',
'Versicolor', 'Virginica',
'Setosa'])

# Encode: Setosa -> 1, Others -> 0
mask = np.where(species ==
'Setosa', 1, 0)

print("Encoded Mask:")
print(mask)
```

```
Encoded Mask:
[1 0 0 1]
```

```
import numpy as np
from sklearn.datasets import
load_iris

# Load Iris dataset
iris = load_iris()
data = iris.data.copy()    # Create
a copy of the Iris array
```

```
# Manually insert NaN at index [0,
0]
data[0, 0] = np.nan

# Locate missing value using
np.isnan()
missing_mask = np.isnan(data)

print("Location of missing
value:")
print(missing_mask)

# Compute mean of the affected
column (column 0), ignoring NaN
col_mean = np.nanmean(data[:, 0])

# Replace NaN with column mean
data[missing_mask] = col_mean

print("\nUpdated data (NaN
replaced with column mean):")
print(data[:5])  # show first 5
rows for verification
```

```
Location of missing value:
[[ True False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
 [False False False False]
...
 [4.9       3.        1.4       0.2      ]
 [4.7       3.2       1.3       0.2      ]
 [4.6       3.1       1.5       0.2      ]
 [5.        3.6       1.4       0.2      ]]
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...