

MACHINE LEARNING ASSIGNMENT

Saipoojitha
2311cs020465 - omega

1. Given a list of numbers with some missing values represented as None, replace the missing values with the mean of the non-missing numbers.

Example Input:

[10, None, 30, None, 50]

Solution:

```
def replace_missing_with_mean(numbers):  
    # Calculate the mean of non-missing numbers  
    non_missing_numbers = [num for num in numbers if num is not None]  
    mean = sum(non_missing_numbers) / len(non_missing_numbers)  
  
    # Replace None values with the mean  
    return [mean if num is None else num for num in numbers]
```

Example Input

```
input_list = [10, None, 30, None, 50]  
output_list = replace_missing_with_mean(input_list)  
print("Output:", output_list)
```

```
1 def replace_missing_with_mean(numbers):  
2     # Calculate the mean of non-missing numbers  
3     non_missing_numbers = [num for num in numbers if num is not None]  
4     mean = sum(non_missing_numbers) / len(non_missing_numbers)  
5  
6     # Replace None values with the mean  
7     return [mean if num is None else num for num in numbers]  
8  
9 # Example Input  
10 input_list = [10, None, 30, None, 50]  
11 output_list = replace_missing_with_mean(input_list)  
12 print("Output:", output_list)  
13
```

Output: **Finished**
Finished in 46 ms
Output: [10, 30.0, 30, 30.0, 50]

2.Implement a function to scale a list of numbers using Min-Max Scaling. Use the formula:

$$\text{scaled}(x) = \frac{x - \min}{\max - \min}$$

Example Input:

[20, 40, 60, 80, 100]

Solution:

```
def scaling(a):  
    h=[]  
    m = max(a)  
    n = min(a)  
    for i in a:  
        a = (i-n)/(m-n)  
        if m==n:  
            h.append(0)  
        else:  
            h.append(a)  
    return h  
a = list(map(int,input().split(" ")))  
print("list:",a)  
o = scaling(a)  
print("Scaled values: ",o)
```

The screenshot shows a web browser window with a code editor. The code is written in Python and implements the Min-Max Scaling function. The code is as follows:

```
1 def scaling(a):  
2     h=[]  
3     m = max(a)  
4     n = min(a)  
5     for i in a:  
6         a = (i-n)/(m-n)  
7         if m==n:  
8             h.append(0)  
9         else:  
10            h.append(a)  
11    return h  
12 a = list(map(int,input().split(" ")))  
13 print("list:",a)  
14 o = scaling(a)  
15 print("Scaled values: ",o)
```

The code is saved as 'ml2'. The output area is empty. The browser window shows the URL 'leetcode.com/playground/UpqjoM3n' and the page title 'ml2 - LeetCode Playground'.

3.Implement a function to binarize a list of numbers given a threshold. All values greater than or equal to the threshold should become 1, and all others should become 0.

Example Input:

[1.5, 2.3, 0.8, 3.0], threshold = 2.0

Solution:

```
def binarize_list(numbers, threshold):  
    # Replace each number with 1 if it is >= threshold, otherwise 0  
    return [1 if num >= threshold else 0 for num in numbers]
```

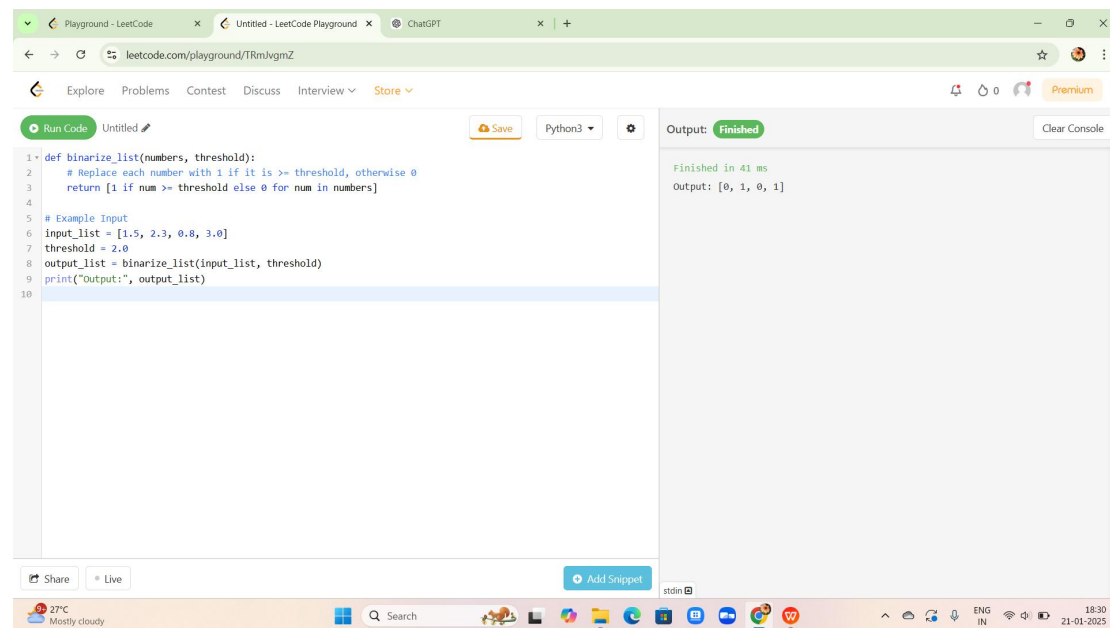
Example Input

```
input_list = [1.5, 2.3, 0.8, 3.0]
```

```
threshold = 2.0
```

```
output_list = binarize_list(input_list, threshold)
```

```
print("Output:", output_list)
```

A screenshot of a web browser window displaying the LeetCode Playground interface. The browser has three tabs: 'Playground - LeetCode', 'Untitled - LeetCode Playground', and 'ChatGPT'. The address bar shows 'leetcode.com/playground/TRmJvgmZ'. The playground interface includes a 'Run Code' button, a 'Save' button, and a language selector set to 'Python3'. The code editor contains the following Python code:

```
1 def binarize_list(numbers, threshold):  
2     # Replace each number with 1 if it is >= threshold, otherwise 0  
3     return [1 if num >= threshold else 0 for num in numbers]  
4  
5 # Example Input  
6 input_list = [1.5, 2.3, 0.8, 3.0]  
7 threshold = 2.0  
8 output_list = binarize_list(input_list, threshold)  
9 print("Output:", output_list)  
10
```

The output panel on the right shows 'Output: Finished' and 'Finished in 41 ms'. Below this, it displays 'Output: [0, 1, 0, 1]'. The bottom of the screen shows a Windows taskbar with various icons and a system clock indicating 18:30 on 21-01-2023.

4. Given a function $f(x)=2x+3$, write a function to approximate $f(x)$ by calculating the outputs for a given list of inputs.

Example Input:

```
inputs = [1, 2, 3]
```

Solution:

```
def f(x):  
    a=[]  
    for i in x:  
        b = 2*i+3  
        a.append(b)  
    return a  
l = list(map(int,input("Enter list elements: ").split(" ")))  
print(l)  
f = f(l)  
print("2x+3 evaluation: ",f)
```

The screenshot shows a web browser window with the LeetCode Playground interface. The URL is `leetcode.com/playground/N77PLGxb`. The interface includes a top navigation bar with links like 'Explore', 'Problems', 'Contest', 'Discuss', 'Interview', and 'Store'. Below this, there's a 'Run Code' button and a 'Save' button. The code editor contains the following Python code:

```
1 def f(x):
2     a=[]
3     for i in x:
4         b = 2*i+3
5         a.append(b)
6     return a
7 l = list(map(int,input("Enter list elements: ").split(" ")))
8 print(l)
9 f = f(l)
10 print("2x+3 evaluation: ",f)
11
12
13
```

The output panel on the right shows the result of running the code:

```
Finished in 47 ms
Enter list elements: [1, 2, 3]
2x+3 evaluation: [5, 7, 9]
```

The bottom of the browser window shows a Windows taskbar with various icons and a system clock indicating 18:32 on 21-01-2025.

5. Write a function to standardize a list of numbers by subtracting the mean and dividing by the standard deviation.

Example Input:

[10, 20, 30, 40]

Solution:

```
def standardize(numbers):
```

```
    # Calculate the mean and standard deviation
```

```
    mean = sum(numbers) / len(numbers)
```

```
    variance = sum((x - mean) ** 2 for x in numbers) / len(numbers)
```

```
    std_dev = variance ** 0.5
```

```
    # Standardize each number
```

```
    return [(x - mean) / std_dev for x in numbers]
```

```
# Example Input
```

```
input_list = [10, 20, 30, 40]
```

```
output_list = standardize(input_list)
```

```
print("Output:", output_list)
```

```
1 def standardize(numbers):
2     # Calculate the mean and standard deviation
3     mean = sum(numbers) / len(numbers)
4     variance = sum((x - mean) ** 2 for x in numbers) / len(numbers)
5     std_dev = variance ** 0.5
6
7     # Standardize each number
8     return [(x - mean) / std_dev for x in numbers]
9
10 # Example Input
11 input_list = [10, 20, 30, 40]
12 output_list = standardize(input_list)
13 print("Output:", output_list)
14
```

Output: Finished

Finished in 48 ms

Output: [-1.3416407864998738, -0.4472135954999579, 0.4472135954999579, 1.3416407864998738]

6. Given a list of data points and their corresponding labels, create a concept representation as a dictionary where keys are unique labels and values are lists of points belonging to each label.

Example Input:

data = [1, 2, 3, 4, 5], labels = ["A", "B", "A", "B", "A"]

Solution:

```
def create_label_representation(data, labels):
    # Create an empty dictionary to store the result
    label_dict = {}

    # Iterate over data and labels
    for point, label in zip(data, labels):
        # Add the point to the corresponding label's list
        if label not in label_dict:
            label_dict[label] = []
        label_dict[label].append(point)

    return label_dict

# Example Input
data = [1, 2, 3, 4, 5]
labels = ["A", "B", "A", "B", "A"]
output = create_label_representation(data, labels)
print("Output:", output)
```

The screenshot shows a web browser window with the LeetCode Playground interface. The URL is `leetcode.com/playground/5Dah2G5Z`. The interface includes a top navigation bar with links like 'Explore', 'Problems', 'Contest', 'Discuss', 'Interview', and 'Store'. Below this, there's a 'Run Code' button and a 'Save' button. The main area is split into two panels: a code editor on the left and an output console on the right. The code editor contains a Python function `create_label_representation` that takes `data` and `labels` as input and returns a dictionary. The output console shows the result of running the code: `output: {'A': [1, 3, 5], 'B': [2, 4]}`. The bottom of the browser window shows a Windows taskbar with various icons and the system clock.

```
1 def create_label_representation(data, labels):
2     # Create an empty dictionary to store the result
3     label_dict = {}
4
5     # Iterate over data and labels
6     for point, label in zip(data, labels):
7         # Add the point to the corresponding label's list
8         if label not in label_dict:
9             label_dict[label] = []
10            label_dict[label].append(point)
11
12    return label_dict
13
14    # Example Input
15    data = [1, 2, 3, 4, 5]
16    labels = ["A", "B", "A", "B", "A"]
17    output = create_label_representation(data, labels)
18    print("Output:", output)
19
```

Output: Finished
Finished in 45 ms
output: {'A': [1, 3, 5], 'B': [2, 4]}

7. Write a function that categorizes a given machine learning task based on its description. The categories are "Supervised Learning" or "Unsupervised Learning"

Example Input:

"Predict the price of a house based on features like size, location, and age."

Solution:

```
def categorize_ml_task(description):
    # Keywords indicating supervised learning
    supervised_keywords = ["predict", "classification", "regression", "label", "target",
                           "output"]

    # Keywords indicating unsupervised learning
    unsupervised_keywords = ["cluster", "group", "pattern", "anomaly detection",
                              "dimension reduction"]

    # Convert description to lowercase for case-insensitive matching
    description = description.lower()

    # Check for supervised learning keywords
    if any(keyword in description for keyword in supervised_keywords):
        return "Supervised Learning"

    # Check for unsupervised learning keywords
    elif any(keyword in description for keyword in unsupervised_keywords):
        return "Unsupervised Learning"

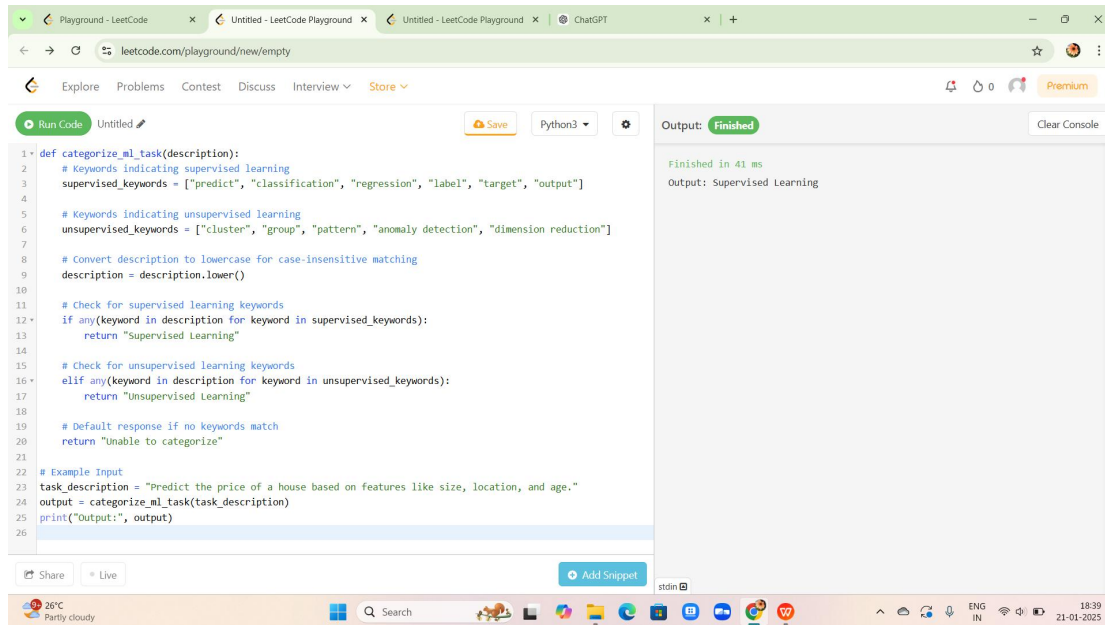
    # Default response if no keywords match
    return "Unable to categorize"
```

Example Input

task_description = "Predict the price of a house based on features like size, location, and age."

output = categorize_ml_task(task_description)

print("Output:", output)



The screenshot shows a web browser window with the LeetCode Playground interface. The URL is leetcode.com/playground/new/empty. The interface includes a sidebar with navigation links (Explore, Problems, Contest, Discuss, Interview, Store) and a main editor area. The editor contains a Python script that defines a function `categorize_ml_task` to categorize machine learning tasks based on keywords. The script includes comments for supervised and unsupervised learning keywords, a function to convert the description to lowercase, and logic to check for keywords and return the appropriate category. An example input is provided at the bottom of the script. The output panel on the right shows the result: "Output: Supervised Learning".

```
1 def categorize_ml_task(description):
2     # Keywords indicating supervised learning
3     supervised_keywords = ["predict", "classification", "regression", "label", "target", "output"]
4
5     # Keywords indicating unsupervised learning
6     unsupervised_keywords = ["cluster", "group", "pattern", "anomaly detection", "dimension reduction"]
7
8     # Convert description to lowercase for case-insensitive matching
9     description = description.lower()
10
11    # Check for supervised learning keywords
12    if any(keyword in description for keyword in supervised_keywords):
13        return "Supervised Learning"
14
15    # Check for unsupervised learning keywords
16    elif any(keyword in description for keyword in unsupervised_keywords):
17        return "Unsupervised Learning"
18
19    # Default response if no keywords match
20    return "Unable to categorize"
21
22    # Example Input
23    task_description = "Predict the price of a house based on features like size, location, and age."
24    output = categorize_ml_task(task_description)
25    print("Output:", output)
26
```

8. Write a function that categorizes a given machine learning task based on its description. The categories are "Supervised Learning" or "Unsupervised Learning"

"Group customers based on their purchasing patterns."

Solution:

```
def categorize_ml_task(description):
```

```
    # Keywords indicating supervised learning
    supervised_keywords = ["predict", "classification", "regression", "label", "target",
"output"]
```

```
    # Keywords indicating unsupervised learning
    unsupervised_keywords = ["cluster", "group", "pattern", "anomaly detection",
"dimension reduction"]
```

```
    # Convert description to lowercase for case-insensitive matching
    description = description.lower()
```

```
    # Check for supervised learning keywords
    if any(keyword in description for keyword in supervised_keywords):
        return "Supervised Learning"
```

```
    # Check for unsupervised learning keywords
```

```

elif any(keyword in description for keyword in unsupervised_keywords):
    return "Unsupervised Learning"

# Default response if no keywords match
return "Unable to categorize"

# Example Input
task_description = "Group customers based on their purchasing patterns."
output = categorize_ml_task(task_description)
print("Output:", output)

```

The screenshot shows a web browser window with the LeetCode Playground interface. The code editor on the left contains a Python function `categorize_ml_task` that checks for keywords in a task description to determine if it's supervised or unsupervised learning. The output panel on the right shows the result: "Output: Unsupervised Learning".

```

1 def categorize_ml_task(description):
2     # Keywords indicating supervised learning
3     supervised_keywords = ["predict", "classification", "regression", "label", "target", "output"]
4
5     # Keywords indicating unsupervised learning
6     unsupervised_keywords = ["cluster", "group", "pattern", "anomaly detection", "dimension reduction"]
7
8     # Convert description to lowercase for case-insensitive matching
9     description = description.lower()
10
11     # Check for supervised learning keywords
12     if any(keyword in description for keyword in supervised_keywords):
13         return "Supervised Learning"
14
15     # Check for unsupervised learning keywords
16     elif any(keyword in description for keyword in unsupervised_keywords):
17         return "Unsupervised Learning"
18
19     # Default response if no keywords match
20     return "Unable to categorize"
21
22 # Example Input
23 task_description = "Group customers based on their purchasing patterns."
24 output = categorize_ml_task(task_description)
25 print("Output:", output)
26

```

Output: Finished
Finished in 37 ms
Output: Unsupervised Learning

9. Predict Y using a Simple Linear Model

Task:

Write a function that calculates the predicted y value given the slope m, intercept c, and an input x using the formula for a line:

$$y = mx + c$$

Input:

Three values:

m (float): The slope of the line.

c (float): The y-intercept of the line.

x (float): The input value.

Output: A single float value representing y.

Solution:

```

def predict_y(m, c, x):
    """

```

Calculate the predicted y value for a simple linear model.

Parameters:

m (float): Slope of the line.
c (float): Intercept of the line.
x (float): Input value.

Returns:

float: The predicted y value.

"""

return m * x + c

Example Input

m = 2.5

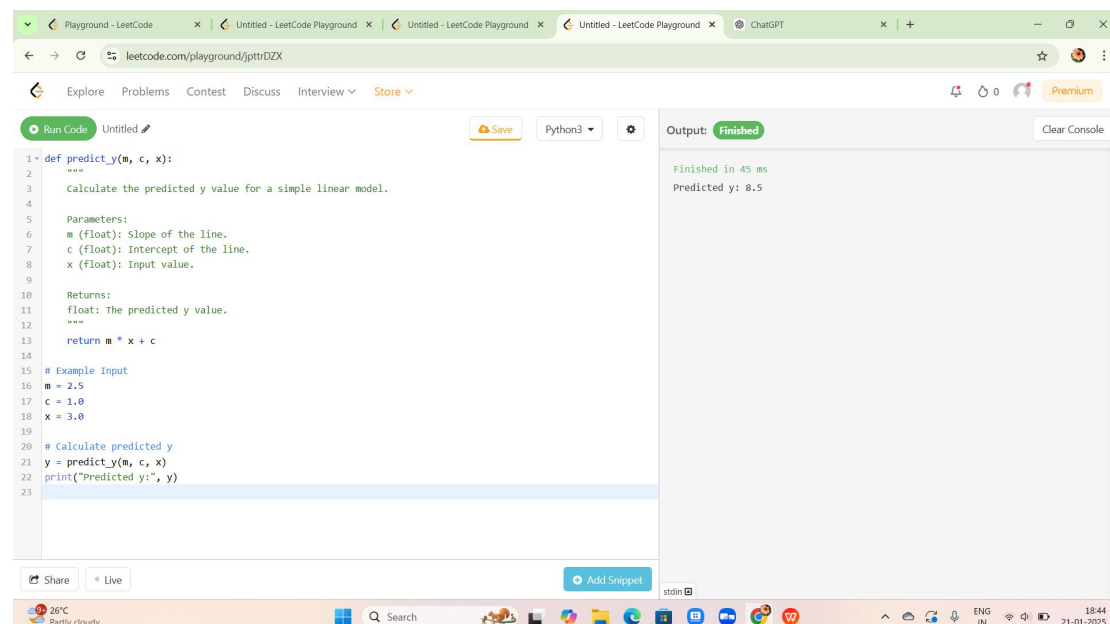
c = 1.0

x = 3.0

Calculate predicted y

y = predict_y(m, c, x)

print("Predicted y:", y)



The screenshot shows a web browser with multiple tabs, including 'Playground - LeetCode', 'Untitled - LeetCode Playground', and 'ChatGPT'. The active tab is 'leetcode.com/playground/jptrDZX'. The code editor displays a Python function `predict_y(m, c, x)` with docstrings for parameters and returns. Below the function, example input values are provided: `m = 2.5`, `c = 1.0`, and `x = 3.0`. The function is then called: `y = predict_y(m, c, x)` and the result is printed: `print("Predicted y:", y)`. The output panel on the right shows 'Finished' and 'Predicted y: 8.5'. The bottom of the screen shows a Windows taskbar with various icons and the system clock indicating 18:44 on 21-01-2025.

10. Given an array of integers representing raw data, remove duplicates and sort the data in ascending order. Write a function to accomplish this.

Example Input:

[4, 2, 2, 8, 3, 3, 1]

Solution:

def remove_duplicates_and_sort(data):

"""

Remove duplicates from the array and sort in ascending order.

Parameters:

data (list of int): Array of integers.

Returns:

list of int: Processed array with unique elements in ascending order.

"""

return sorted(set(data))

Example Input

input_data = [4, 2, 2, 8, 3, 3, 1]

Process the data

output_data = remove_duplicates_and_sort(input_data)

print("Output:", output_data)

The screenshot shows a web browser window with the LeetCode Playground interface. The URL is `leetcode.com/playground/jpttrDZX`. The interface includes a top navigation bar with links for Explore, Problems, Contest, Discuss, Interview, and Store. Below this is a toolbar with buttons for Run Code, Save, Python3, and a settings icon. The main area is split into two panels. The left panel contains the following Python code:

```
1 def remove_duplicates_and_sort(data):
2     """
3     Remove duplicates from the array and sort in ascending order.
4
5     Parameters:
6     data (list of int): Array of integers.
7
8     Returns:
9     list of int: Processed array with unique elements in ascending order.
10    """
11    return sorted(set(data))
12
13 # Example Input
14 input_data = [4, 2, 2, 8, 3, 3, 1]
15
16 # Process the data
17 output_data = remove_duplicates_and_sort(input_data)
18 print("Output:", output_data)
19
```

The right panel shows the output of the code execution. It displays "Finished" in green, followed by "Finished in 45 ms" and "Predicted y: 8.5". Below this, it shows "Finished in 46 ms" and "Output: [1, 2, 3, 4, 8]". At the bottom of the browser window, a Windows taskbar is visible with the date and time set to 18:46 on 21-01-2023.