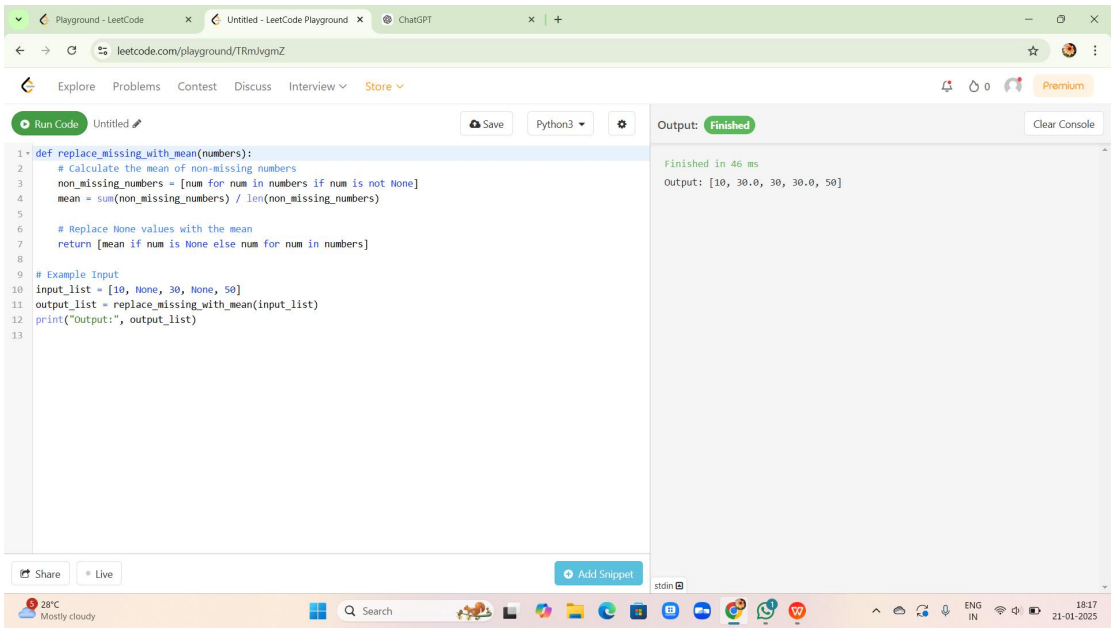
MACHINE LEARNING ASSIGNMENT

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)
```



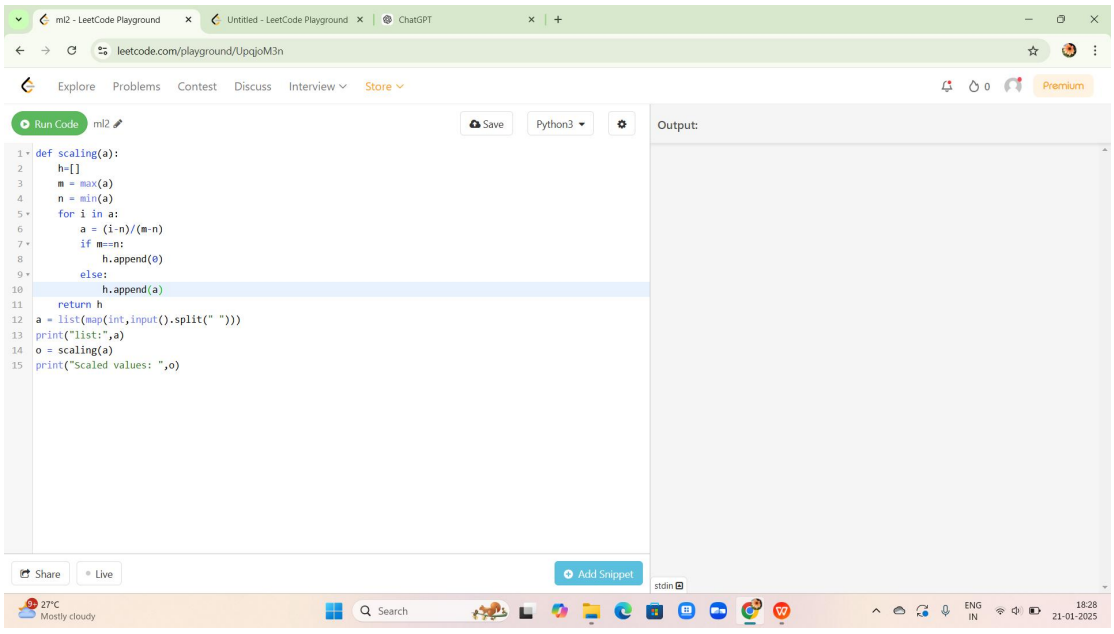
2.Implement a function to scale a list of numbers using Min-Max Scaling. Use the formula:

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

Example Input:
[20, 40, 60, 80, 100]

Solution:

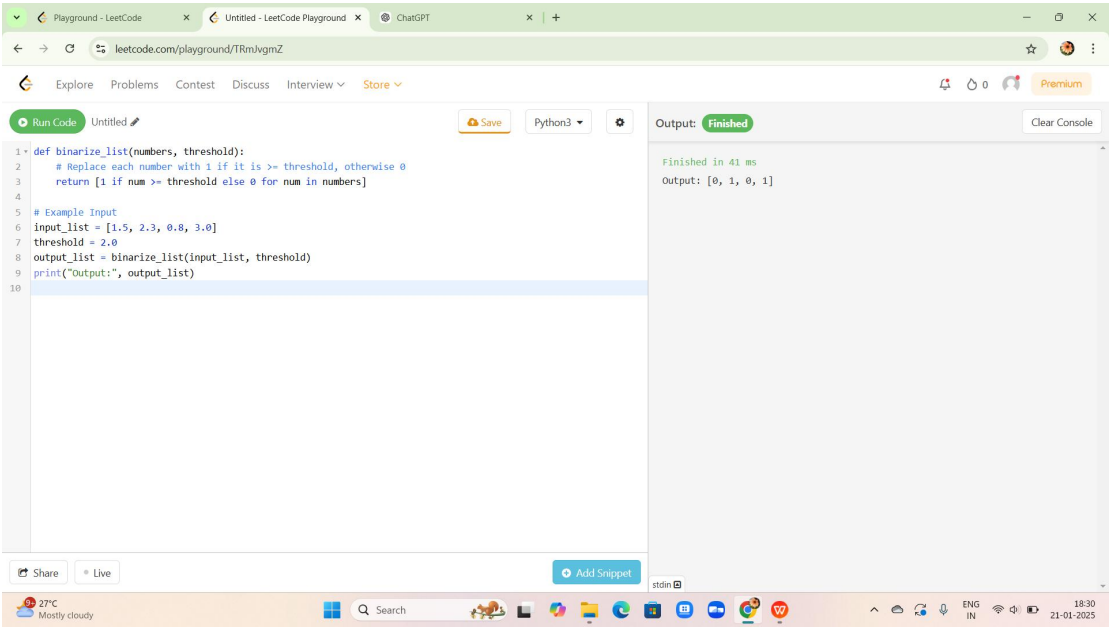
```
def scaling(a):
    h=[]
    m = max(a)
    n = min(a)
    for i in a:
        a = (i-n)/(m-n)
        if m==n:
            h.append(0)
        else:
            h.append(a)
    return h
a = list(map(int,input().split(" ")))
print("list:",a)
o = scaling(a)
print("Scaled values: ",o)
```



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:

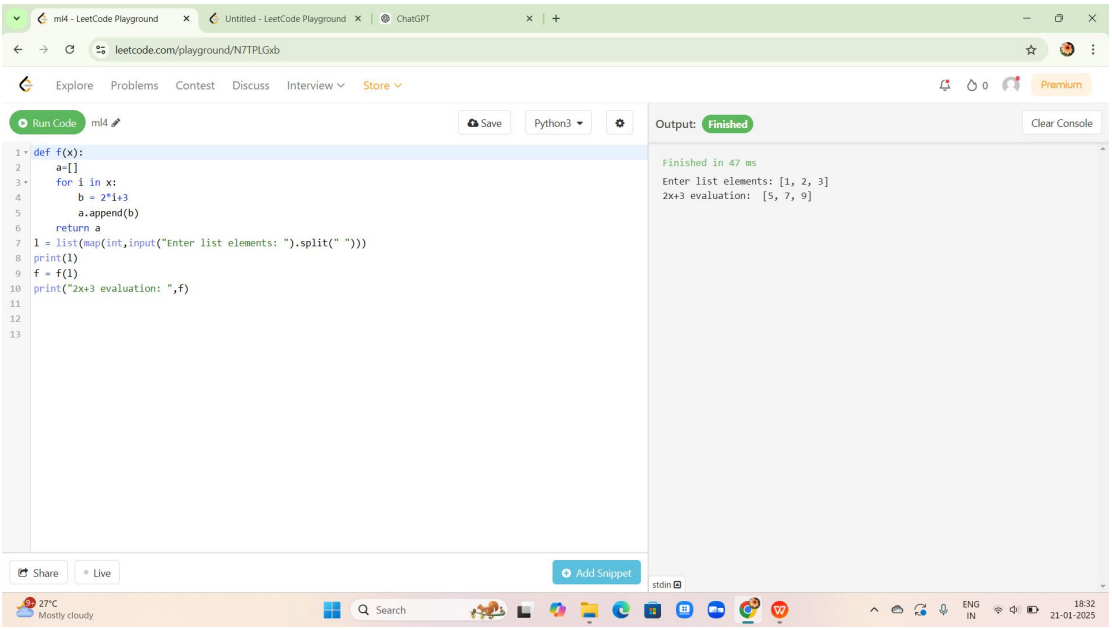
```
def binarize_list(numbers, threshold):
    # Replace each number with 1 if it is >= threshold, otherwise 0
    return [1 if num >= threshold else 0 for num in numbers]

# 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)
```



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)
```



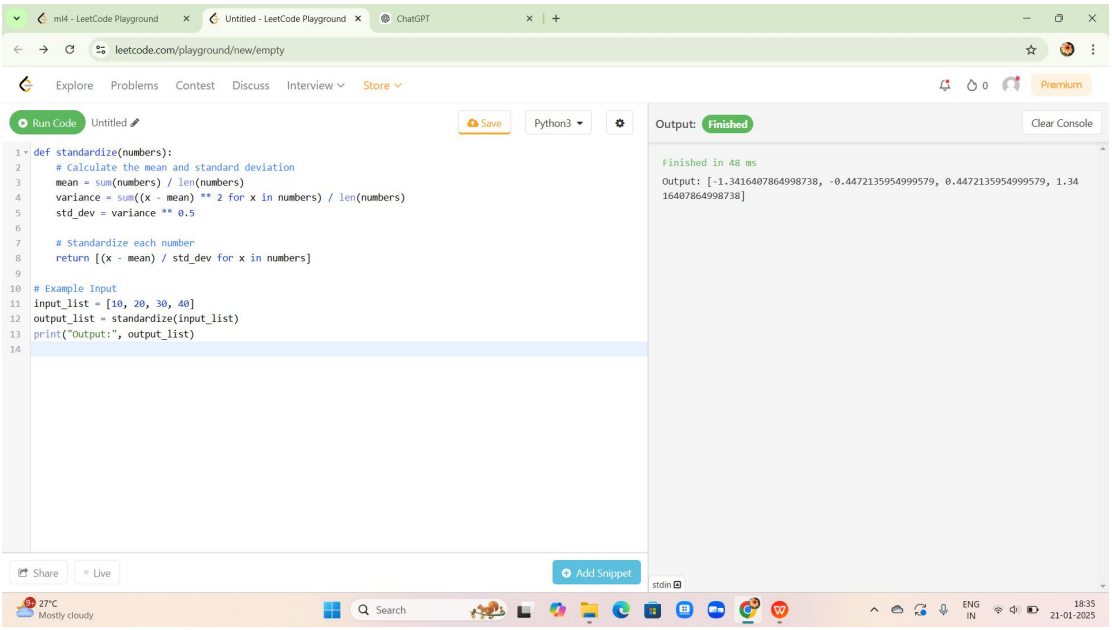
5.Write a function to standardize a list of numbers by subtracting the mean and dividing by the standard deviation.

Example Input:

[10, 20, 30, 40]

Solution:

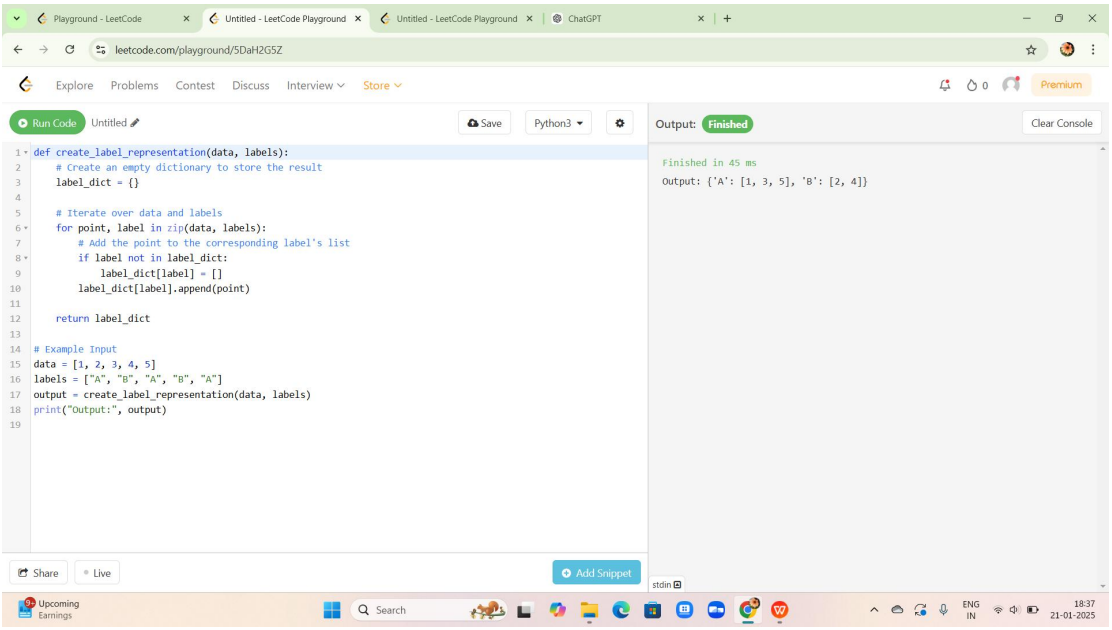
```
def standardize(numbers):
    # Calculate the mean and standard deviation
    mean = sum(numbers) / len(numbers)
    variance = sum((x - mean) ** 2 for x in numbers) / len(numbers)
    std_dev = variance ** 0.5

    # 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)
```



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)
```

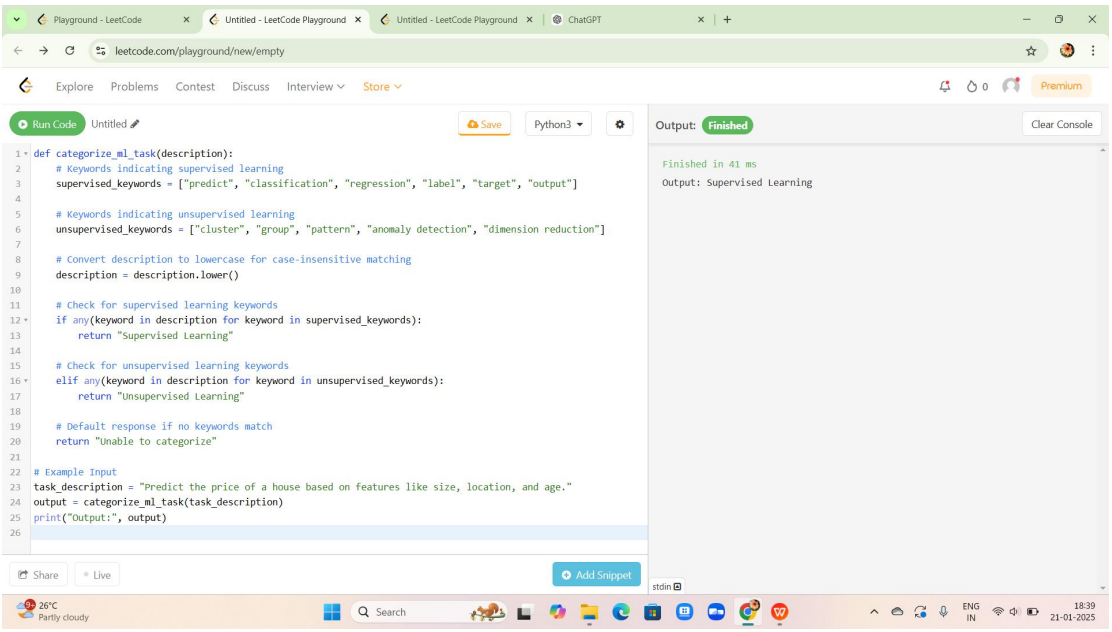


7.Write a function that categorizes a given machine learning task based on its description. The categories are "Supervised Learning" or "Unsupervised Learning"
Example Input:
"Predict the price of a house based on features like size, location, and age."

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)
```

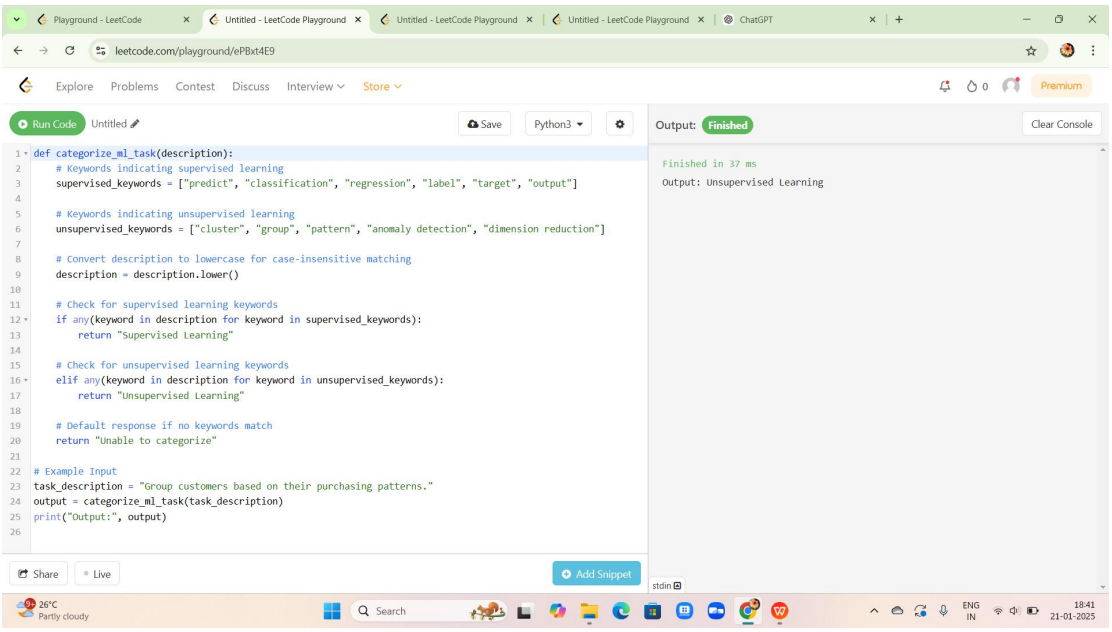


8.Write a function that categorizes a given machine learning task based on its description. The categories are "Supervised Learning" or "Unsupervised Learning"
"Group customers based on their purchasing patterns."

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)
```

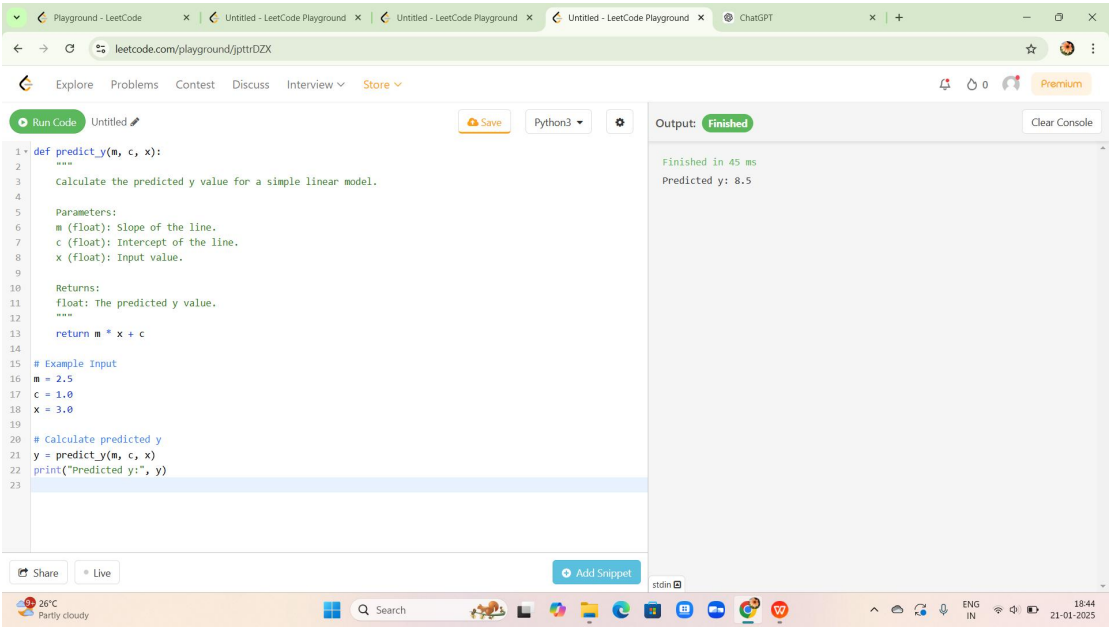


9. Predict Y using a Simple Linear Model
Task:
Write a function that calculates the predicted y value given the slope m, intercept c,
and an input x using the formula for a line:
y=mx+c
Input:
Three values:
m (float): The slope of the line.
c (float): The y-intercept of the line.
x (float): The input 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)
```



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)
```