FDS Lab-2 Assignment

`Name` : `Ayush Panchal`

`Roll No.` : `P24DS013`

1. Create arrays of one 1D with 20 numbers and one 2D of 10 x 10. Assign random numbers between 1 to 100 to the matrix

In [ ]:
```
size_1d = 20
size_2d = [10, 10]
```

Linear Congruential Generator formula: `X_{n+1} = (a * X_n + c) % m`

In [ ]:
```
class SimpleRandom:
    def __init__(self, seed=1):
        self.modulus = 2**31 - 1

        # Multiplier. 48271 best for generating good randomness
        self.a = 48271

        # Increment
        self.c = 0

        # Seed or initial state
        self.state = seed

    def random(self):
        self.state = (self.a * self.state + self.c) % self.modulus
        # Scale the result to the range [0, 100]
        return int((self.state / self.modulus) * 100)
```

In [ ]:
```
one_d_array = []

random_num = SimpleRandom(seed=1456)
for i in range(size_1d):
    one_d_array.append(random_num.random())

one_d_array
```

Out[ ]:
```
[3, 80, 56, 18, 34, 18, 80, 50, 79, 55, 38, 92, 72, 72, 83, 5, 37, 84, 36, 38]
```

In [ ]:
```
two_d_array = []

random = SimpleRandom(seed=12456)
for i in range(size_2d[0]):

    temp_arr = []
    for j in range(size_2d[1]):
        temp_arr.append(random.random())
    two_d_array.append(temp_arr)
```

```
two_d_array
```

Out[ ]:
```
[[27, 16, 44, 91, 85, 77, 53, 59, 75, 19],
 [40, 21, 25, 96, 94, 89, 35, 77, 69, 11],
 [70, 88, 14, 75, 51, 85, 66, 9, 65, 38],
 [85, 81, 13, 50, 47, 55, 94, 46, 2, 25],
 [74, 22, 98, 41, 11, 57, 62, 79, 26, 24],
 [89, 81, 57, 78, 86, 82, 23, 68, 12, 15],
 [83, 81, 78, 86, 22, 37, 29, 86, 77, 60],
 [85, 11, 22, 74, 53, 85, 14, 13, 39, 20],
 [65, 45, 55, 68, 75, 56, 55, 46, 65, 45],
 [58, 97, 69, 54, 67, 11, 16, 90, 52, 12]]
```

2. Display the total number of elements in both arrays.

In [ ]:
```
count = 0
for i in one_d_array:
    count+=1

f"Total Numbers of element in 1D Array = {count}"
```

Out[ ]:
```
'Total Numbers of element in 1D Array = 20'
```

In [ ]:
```
count = 0
for i in two_d_array:
    for j in i:
        count += 1
f"Total Numbers of element in 2D Array = {count}"
```

Out[ ]:
```
'Total Numbers of element in 2D Array = 100'
```

3. Sort the numbers in ascending order for both arrays.

In [ ]:
```
# Function to find the partition position
def partition(array, low, high):
    # Choose the rightmost element as pivot
    pivot = array[high]

    # Pointer for greater element
    i = low - 1

    # Traverse through all elements
    # Compare each element with pivot
    for j in range(low, high):
        if array[j] <= pivot:
            # If element smaller than pivot is found
            # swap it with the greater element pointed by i
            i = i + 1
            # Swapping element at i with element at j
            array[i], array[j] = array[j], array[i]

    # Swap the pivot element with the greater element specified by i
    array[i + 1], array[high] = array[high], array[i + 1]
```

```
    # Return the position from where partition is done
    return i + 1

# Function to perform quicksort
def quickSort(array, low, high):
    if low < high:
        # Find pivot element such that
        # element smaller than pivot are on the left
        # element greater than pivot are on the right
        pi = partition(array, low, high)

        # Recursive call on the left of pivot
        quickSort(array, low, pi - 1)

        # Recursive call on the right of pivot
        quickSort(array, pi + 1, high)

    return array

arr = [10, 7, 8, 9, 1, 5]
sorted_array = quickSort(arr, 0, len(arr) - 1)
print("Sorted array:", sorted_array)
```

```
Sorted array: [1, 5, 7, 8, 9, 10]
```

In [ ]:

```
sorted_1d_array = quickSort(one_d_array.copy(), 0, len(one_d_array) - 1)
sorted_1d_array
```

Out[ ]:

```
[3, 5, 18, 18, 34, 36, 37, 38, 38, 50, 55, 56, 72, 72, 79, 80, 80, 83, 84, 92]
```

In [ ]:

```
partially_sorted_2d_array = []
for i in two_d_array:
    sorted_sub_array = quickSort(i, 0, len(i)-1)
    for j in sorted_sub_array:
        partially_sorted_2d_array.append(j)

print(partially_sorted_2d_array)
sorted_2d_to_1d_array = quickSort(partially_sorted_2d_array, 0, len(partially_sc
```

```
[16, 19, 27, 44, 53, 59, 75, 77, 85, 91, 11, 21, 25, 35, 40, 69, 77, 89, 94, 96,
9, 14, 38, 51, 65, 66, 70, 75, 85, 88, 2, 13, 25, 46, 47, 50, 55, 81, 85, 94, 11,
22, 24, 26, 41, 57, 62, 74, 79, 98, 12, 15, 23, 57, 68, 78, 81, 82, 86, 89, 22, 2
9, 37, 60, 77, 78, 81, 83, 86, 86, 11, 13, 14, 20, 22, 39, 53, 74, 85, 85, 45, 4
5, 46, 55, 55, 56, 65, 65, 68, 75, 11, 12, 16, 52, 54, 58, 67, 69, 90, 97]
```

In [ ]:

```
sorted_2d_array = []
partition_size = size_2d[0]
count = 0
for i in range(partition_size):
    sub_array = []
    for j in range(size_2d[1]):
        sub_array.append(partially_sorted_2d_array[count])
        count+=1
    sorted_2d_array.append(sub_array)

sorted_2d_array
```

```
Out[ ]: [[2, 9, 11, 11, 11, 11, 12, 12, 13, 13],
 [14, 14, 15, 16, 16, 19, 20, 21, 22, 22],
 [22, 23, 24, 25, 25, 26, 27, 29, 35, 37],
 [38, 39, 40, 41, 44, 45, 45, 46, 46, 47],
 [50, 51, 52, 53, 53, 54, 55, 55, 55, 56],
 [57, 57, 58, 59, 60, 62, 65, 65, 65, 66],
 [67, 68, 68, 69, 69, 70, 74, 74, 75, 75],
 [75, 77, 77, 77, 78, 78, 79, 81, 81, 81],
 [82, 83, 85, 85, 85, 85, 85, 86, 86, 86],
 [88, 89, 89, 90, 91, 94, 94, 96, 97, 98]]
```

4. Perform all arithmetic and statistical operations. Display the results. (Operations should include: min, max, count, sum, mean, median, mode, variance, standard deviation)

```
In [ ]: #Min and max
min = 101
max = -1

for i in one_d_array:

    if i<min:
        min = i

    if i>max:
        max = i

print(f"Minimum Element : {min}")
print(f"Minimum Element : {max}")

#Count
count_map = {}
for i in one_d_array:
    if i not in count_map.keys():
        count_map[i] = 1
    else:
        count_map[i] += 1

print()
print(count_map)

# Sum
sum = 0
for i in one_d_array:
    sum += i

print()
print(f"Sum of Array : {sum}")

# Mean
sum = 0
for i in one_d_array:
    sum+=i
mean = sum / len(one_d_array)

print()
```

```python
print(f"Mean of array : {mean}")

# Median

middle_element_index = (len(one_d_array) // 2)
print()
print(f"Median Element : {sorted_1d_array[middle_element_index]}")

# Mode
highest_count = 0
highest_count_key = -1
for i in count_map.keys():
    if count_map[i] > highest_count:
        highest_count = count_map[i]
        highest_count_key = i

print()
print(f"Mode : {highest_count_key}")

# Variance
sum = 0
for i in one_d_array:
    sum += (i - mean)**2
variance = sum / len(one_d_array)

print()
print(f"Variance : {variance}")

# Standard Deviation
sd = variance**(1/2)

print()
print(f"Stabndard Deviation : {sd}")
```

```
Minimum Element : 3
Minimum Element : 92

{3: 1, 80: 2, 56: 1, 18: 2, 34: 1, 50: 1, 79: 1, 55: 1, 38: 2, 92: 1, 72: 2, 83: 1, 5: 1, 37: 1, 84: 1, 36: 1}

Sum of Array : 1030

Mean of array : 51.5

Median Element : 55

Mode : 80

Variance : 741.25

Stabndard Deviation : 27.225906780123964
```

In [ ]:

```python
# Assuming 'two_d_array' is a 2D list where each inner list represents a row in

# Number of rows and columns in the 2D array
rows = len(two_d_array)
cols = len(two_d_array[0])

# Min and Max for each column
for col in range(cols):
```

```
    min_val = float('inf')
    max_val = float('-inf')

    for row in range(rows):
        value = two_d_array[row][col]

        if value < min_val:
            min_val = value

        if value > max_val:
            max_val = value

    print(f"Column {col} -> Minimum Element : {min_val}, Maximum Element : {max_

# Count for each column
for col in range(cols):
    count_map = {}

    for row in range(rows):
        value = two_d_array[row][col]

        if value not in count_map:
            count_map[value] = 1
        else:
            count_map[value] += 1

    print(f"\nColumn {col} -> Count Map: {count_map}")

# Sum for each column
for col in range(cols):
    column_sum = 0

    for row in range(rows):
        column_sum += two_d_array[row][col]

    print(f"\nColumn {col} -> Sum: {column_sum}")

# Mean for each column
for col in range(cols):
    column_sum = 0

    for row in range(rows):
        column_sum += two_d_array[row][col]

    column_mean = column_sum / rows
    print(f"\nColumn {col} -> Mean: {column_mean}")

# Median for each column
for col in range(cols):
    column_values = [two_d_array[row][col] for row in range(rows)]
    sorted_column = quickSort(column_values, 0, len(column_values) - 1)
    middle_index = rows // 2

    if rows % 2 == 0:
        median = (sorted_column[middle_index - 1] + sorted_column[middle_index])
    else:
        median = sorted_column[middle_index]

    print(f"\nColumn {col} -> Median: {median}")
```

```python
# Mode for each column
for col in range(cols):
    count_map = {}

    for row in range(rows):
        value = two_d_array[row][col]

        if value not in count_map:
            count_map[value] = 1
        else:
            count_map[value] += 1

    highest_count = 0
    mode_value = None

    for key, count in count_map.items():
        if count > highest_count:
            highest_count = count
            mode_value = key

    print(f"\nColumn {col} -> Mode: {mode_value}")

# Variance for each column
for col in range(cols):
    column_sum = 0

    for row in range(rows):
        column_sum += two_d_array[row][col]

    column_mean = column_sum / rows
    variance_sum = 0

    for row in range(rows):
        variance_sum += (two_d_array[row][col] - column_mean) ** 2

    variance = variance_sum / rows
    print(f"\nColumn {col} -> Variance: {variance}")

# Standard Deviation for each column
for col in range(cols):
    column_sum = 0

    for row in range(rows):
        column_sum += two_d_array[row][col]

    column_mean = column_sum / rows
    variance_sum = 0

    for row in range(rows):
        variance_sum += (two_d_array[row][col] - column_mean) ** 2

    variance = variance_sum / rows
    standard_deviation = variance ** 0.5

    print(f"\nColumn {col} -> Standard Deviation: {standard_deviation}")
```

```
Column 0 -> Minimum Element : 2, Maximum Element : 45
Column 1 -> Minimum Element : 12, Maximum Element : 45
Column 2 -> Minimum Element : 14, Maximum Element : 46
Column 3 -> Minimum Element : 20, Maximum Element : 60
Column 4 -> Minimum Element : 22, Maximum Element : 77
Column 5 -> Minimum Element : 39, Maximum Element : 78
Column 6 -> Minimum Element : 53, Maximum Element : 81
Column 7 -> Minimum Element : 65, Maximum Element : 89
Column 8 -> Minimum Element : 68, Maximum Element : 94
Column 9 -> Minimum Element : 75, Maximum Element : 98

Column 0 -> Count Map: {16: 1, 11: 4, 9: 1, 2: 1, 12: 1, 22: 1, 45: 1}

Column 1 -> Count Map: {19: 1, 21: 1, 14: 1, 13: 2, 22: 1, 15: 1, 29: 1, 45: 1, 1
2: 1}

Column 2 -> Count Map: {27: 1, 25: 2, 38: 1, 24: 1, 23: 1, 37: 1, 14: 1, 46: 1, 1
6: 1}

Column 3 -> Count Map: {44: 1, 35: 1, 51: 1, 46: 1, 26: 1, 57: 1, 60: 1, 20: 1, 5
5: 1, 52: 1}

Column 4 -> Count Map: {53: 1, 40: 1, 65: 1, 47: 1, 41: 1, 68: 1, 77: 1, 22: 1, 5
5: 1, 54: 1}

Column 5 -> Count Map: {59: 1, 69: 1, 66: 1, 50: 1, 57: 1, 78: 2, 39: 1, 56: 1, 5
8: 1}

Column 6 -> Count Map: {75: 1, 77: 1, 70: 1, 55: 1, 62: 1, 81: 2, 53: 1, 65: 1, 6
7: 1}

Column 7 -> Count Map: {77: 1, 89: 1, 75: 1, 81: 1, 74: 2, 82: 1, 83: 1, 65: 1, 6
9: 1}

Column 8 -> Count Map: {85: 4, 94: 1, 79: 1, 86: 2, 68: 1, 90: 1}

Column 9 -> Count Map: {91: 1, 96: 1, 88: 1, 94: 1, 98: 1, 89: 1, 86: 1, 85: 1, 7
5: 1, 97: 1}

Column 0 -> Sum: 150

Column 1 -> Sum: 203

Column 2 -> Sum: 275

Column 3 -> Sum: 446

Column 4 -> Sum: 522

Column 5 -> Sum: 610

Column 6 -> Sum: 686

Column 7 -> Sum: 769

Column 8 -> Sum: 843

Column 9 -> Sum: 899

Column 0 -> Mean: 15.0
```

```
Column 1 -> Mean: 20.3
Column 2 -> Mean: 27.5
Column 3 -> Mean: 44.6
Column 4 -> Mean: 52.2
Column 5 -> Mean: 61.0
Column 6 -> Mean: 68.6
Column 7 -> Mean: 76.9
Column 8 -> Mean: 84.3
Column 9 -> Mean: 89.9
Column 0 -> Median: 11.0
Column 1 -> Median: 17.0
Column 2 -> Median: 25.0
Column 3 -> Median: 48.5
Column 4 -> Median: 53.5
Column 5 -> Median: 58.5
Column 6 -> Median: 68.5
Column 7 -> Median: 76.0
Column 8 -> Median: 85.0
Column 9 -> Median: 90.0
Column 0 -> Mode: 11
Column 1 -> Mode: 13
Column 2 -> Mode: 25
Column 3 -> Mode: 44
Column 4 -> Mode: 53
Column 5 -> Mode: 78
Column 6 -> Mode: 81
Column 7 -> Mode: 74
Column 8 -> Mode: 85
Column 9 -> Mode: 91
Column 0 -> Variance: 122.8
```

```
Column 1 -> Variance: 93.41

Column 2 -> Variance: 90.25

Column 3 -> Variance: 164.04

Column 4 -> Variance: 225.35999999999999

Column 5 -> Variance: 132.6

Column 6 -> Variance: 90.83999999999999

Column 7 -> Variance: 45.08999999999999

Column 8 -> Variance: 42.81

Column 9 -> Variance: 43.69

Column 0 -> Standard Deviation: 11.081516141756055

Column 1 -> Standard Deviation: 9.664884893261792

Column 2 -> Standard Deviation: 9.5

Column 3 -> Standard Deviation: 12.807810117268291

Column 4 -> Standard Deviation: 15.011995203836165

Column 5 -> Standard Deviation: 11.515207336387824

Column 6 -> Standard Deviation: 9.531002045955084

Column 7 -> Standard Deviation: 6.714908785679817

Column 8 -> Standard Deviation: 6.542935121182236

Column 9 -> Standard Deviation: 6.609841147864296
```