```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
#Calculate and interpret mean, median, mode, variance and standard deviation for a given dataset.
Data =[ 15,21,29,21,15,24,32,21,15,30]
print(np.mean(Data))
print(np.median(Data))
print(np.var(Data))
print(np.std(Data))
```

```
22.3
21.0
36.61
6.050619802962338
```

```
#question 3
robo=pd.read_csv('robot_dataset(robot_dataset)_1.csv')
```

```
robo
```

	Robot_ID	Task_Type	Component_ID	Sensor_Type	Sensor_Data	Processing_Time (s)	Accuracy (%)	Environmental_Status	Energy_Consumptio (kWh
0	RBT_001	Inspection	CMP_460	LIDAR	1 (obstacle detected)	67.0	90.4	Stable	2.
1	RBT_002	Assembly	CMP_252	Thermal	85.3 (Â°C)	71.2	98.1	Stable	2.
2	RBT_003	Inspection	CMP_248	Thermal	92% (visual fit)	49.2	95.3	Unstable	2.
3	RBT_004	Welding	CMP_433	Camera	98% (defect-free)	74.5	90.2	Stable	2.
4	RBT_005	Assembly	CMP_992	Camera	92% (visual fit)	64.5	97.2	Unstable	1.
...	...	...	...	...	...	...	...	...	.
495	RBT_496	Inspection	CMP_834	LIDAR	85.3 (Â°C)	66.3	96.2	Unstable	1.
496	RBT_497	Inspection	CMP_851	LIDAR	92% (visual fit)	45.1	92.8	Unstable	2.
497	RBT_498	Inspection	CMP_657	LIDAR	82.4 (Â°C)	75.4	98.7	Unstable	2.
498	RBT_499	Assembly	CMP_562	Camera	98% (defect-free)	48.7	94.9	Stable	2.
499	RBT_500	Assembly	CMP_465	LIDAR + Camera	75.8 (Â°C)	73.5	91.4	Stable	2.

500 rows × 17 columns

```
y=robo['Interaction_Count']
```

```
y.mean()
```

```
5.51
```

```
total_steps=robo['Steps_Walked']
```

```
total_steps.sum()
```

```
14379
```

```
energy=robo['Energy_Consumption (kWh)']
```

```
energy.min()
```

```
1.0
```

```
energy.max()
```

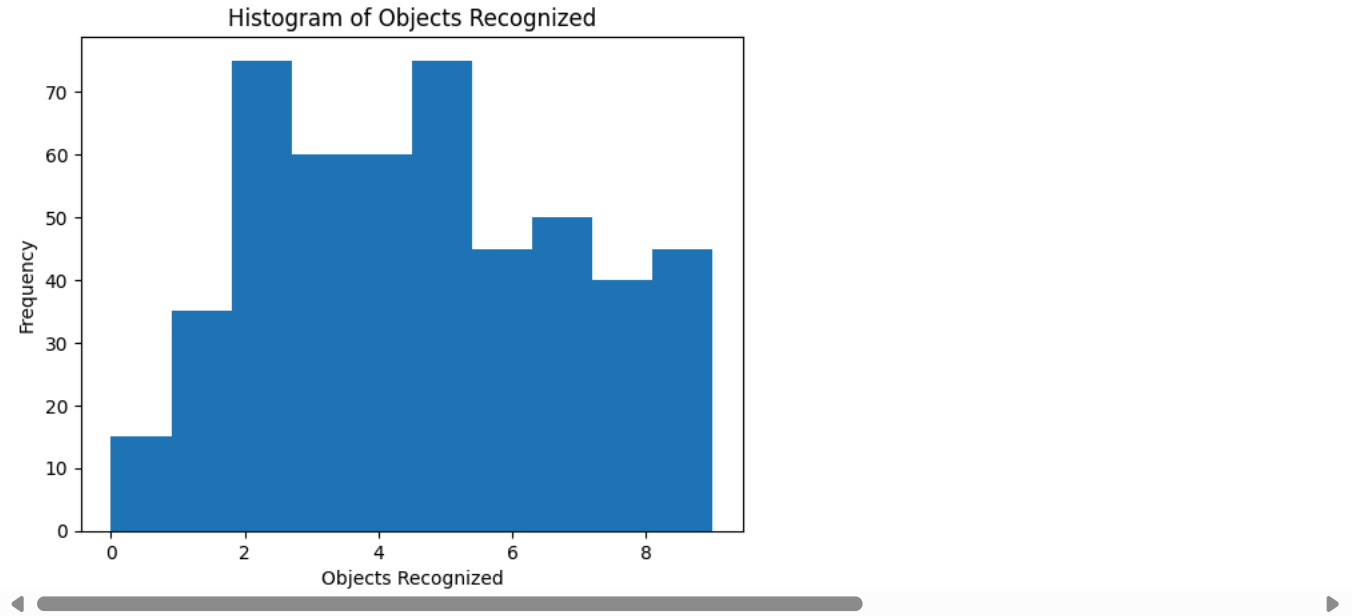
```
3.0
```

```
correlation = robo['Steps_Walked'].corr(robo['Energy_Consumption (kWh)'])


print(correlation)
```

```
0.0015478137393313933
```

```
objects_recognized = np.array(robo['Objects_Recognized'])


plt.hist(objects_recognized)
plt.xlabel('Objects Recognized')
plt.ylabel('Frequency')
plt.title('Histogram of Objects Recognized')
```

```
Text(0.5, 1.0, 'Histogram of Objects Recognized')
```




```
learning_sessions_variance = np.var(robo['Learning_Sessions'])


print(learning_sessions_variance)
```

```
391.15840000000026
```

```
#4 question
name = "Alice"
age = 25
print(f"My name is {name} and I am {age} years old.")
```

```
My name is Alice and I am 25 years old.
```

```
#5th question
num = int(input("Enter an integer: "))
if num > 0:
    print(f"The number {num} is positive.")
elif num < 0:
    print(f"The number {num} is negative.")
else:
    print(f"The number is zero.")
```

```
Enter an integer: 3
The number 3 is positive.
```

```
#6th question
num = int(input("Enter a number: "))
print(f"\nMultiplication Table for {num}:")
for i in range(1, 11):
    print(f"{num} x {i} = {num * i}")
```

```
Enter a number: 4

Multiplication Table for 4:
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
```

```
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40
```

```
#7th question
fruits = ["Apple", "Banana", "Orange", "Grapes", "Mango"]
print("Original list of fruits:", fruits)
fruits.append("Pineapple")
print("After adding Pineapple:", fruits)
fruits.remove("Orange")
print("After removing Orange:", fruits)
fruits.sort()
print("Sorted list:", fruits)
```

```
Original list of fruits: ['Apple', 'Banana', 'Orange', 'Grapes', 'Mango']
After adding Pineapple: ['Apple', 'Banana', 'Orange', 'Grapes', 'Mango', 'Pineapple']
After removing Orange: ['Apple', 'Banana', 'Grapes', 'Mango', 'Pineapple']
Sorted list: ['Apple', 'Banana', 'Grapes', 'Mango', 'Pineapple']
```

```
#8th question
numbers = (10, 20, 30, 40, 50)
print("Original tuple:", numbers)
print("Length of the tuple:", len(numbers))
print("Maximum value:", max(numbers))
print("Minimum value:", min(numbers))
print("Sum of all elements:", sum(numbers))
```

```
Original tuple: (10, 20, 30, 40, 50)
Length of the tuple: 5
Maximum value: 50
Minimum value: 10
Sum of all elements: 150
```

```
#9th question
students = {
    "Alice": 85,
    "Bob": 78,
    "Charlie": 92
}
print("Original dictionary:", students)
students["Bob"] = 88
print("After updating Bob's score:", students)
students["David"] = 95
print("After adding David:", students)
del students["Alice"]
print("After removing Alice:", students)
```

```
Original dictionary: {'Alice': 85, 'Bob': 78, 'Charlie': 92}
After updating Bob's score: {'Alice': 85, 'Bob': 88, 'Charlie': 92}
After adding David: {'Alice': 85, 'Bob': 88, 'Charlie': 92, 'David': 95}
After removing Alice: {'Bob': 88, 'Charlie': 92, 'David': 95}
```

```
#10th question
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
union_set = set1.union(set2)
print("Union of Set 1 and Set 2:", union_set)
intersection_set = set1.intersection(set2)
print("Intersection of Set 1 and Set 2:", intersection_set)
difference_set = set1.difference(set2)
print("Difference between Set 1 and Set 2:", difference_set)
```

```
Union of Set 1 and Set 2: {1, 2, 3, 4, 5, 6, 7, 8}
Intersection of Set 1 and Set 2: {4, 5}
Difference between Set 1 and Set 2: {1, 2, 3}
```

```
#11th question
def find_largest(numbers):
    if not numbers:
        return "The list is empty."
    return max(numbers)



sample_list = [12, 45, 78, 34, 89, 23]

largest_number = find_largest(sample_list)

print(f"The largest number in the list is: {largest_number}")
```

```
The largest number in the list is: 89
```

```
#12th question
squares_of_even = [x**2 for x in range(1, 21) if x % 2 == 0]

print("Squares of even numbers between 1 and 20:", squares_of_even)
```

```
Squares of even numbers between 1 and 20: [4, 16, 36, 64, 100, 144, 196, 256, 324, 400]
```

```
#13th question
product = lambda a, b: a * b

num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

result = product(num1, num2)
print(f"The product of {num1} and {num2} is: {result}")
```

```
Enter the first number: 4
Enter the second number: 5
The product of 4.0 and 5.0 is: 20.0
```

```
#14th question
one_d_array = np.array([1, 2, 3, 4, 5])
two_d_array = np.array([[1, 2, 3], [4, 5, 6]])
three_d_array = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

print("One-dimensional array:", one_d_array)
print("Two-dimensional array:", two_d_array)
print("Three-dimensional array:", three_d_array)

print("One-dimensional array shape:", one_d_array.shape)
print("One-dimensional array dimensions:", one_d_array.ndim)

print("Two-dimensional array shape:", two_d_array.shape)
print("Two-dimensional array dimensions:", two_d_array.ndim)

print("Three-dimensional array shape:", three_d_array.shape)
print("Three-dimensional array dimensions:", three_d_array.ndim)
```

```
One-dimensional array: [1 2 3 4 5]
Two-dimensional array: [[1 2 3]
 [4 5 6]]
Three-dimensional array: [[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
One-dimensional array shape: (5,)
One-dimensional array dimensions: 1
Two-dimensional array shape: (2, 3)
Two-dimensional array dimensions: 2
Three-dimensional array shape: (2, 2, 2)
Three-dimensional array dimensions: 3
```

```
#15th question

random_array = np.random.randint(1, 101, size=(5, 5))

print("5x5 Random Integer Array:")
print(random_array)

print("\nArray Indexing Operations:")

print("Element in the second row, third column:", random_array[1, 2])
print("Entire second row:", random_array[1, :])
print("Entire third column:", random_array[:, 2])
```

```
print("Top-left 2x2 subarray:")
print(random_array[:2, :2])
print("Last two rows and last two columns:")
print(random_array[-2:, -2:])
```

```
5x5 Random Integer Array:
[[ 43  12  96  86  19]
 [ 83  45  26 100 100]
 [ 38  48  66  75  64]
 [ 34  62  40  83  90]
 [ 98   5   8  48   5]]

Array Indexing Operations:
Element in the second row, third column: 26
Entire second row: [ 83  45  26 100 100]
Entire third column: [96 26 66 40  8]
Top-left 2x2 subarray:
[[43 12]
 [83 45]]
Last two rows and last two columns:
[[83 90]
 [48  5]]
```

```
#16th question

array = np.arange(1, 17).reshape(4, 4)

print("4x4 Array with numbers from 1 to 16:")
print(array)

print("\nArray Slicing Operations:")

print("First two rows and first two columns:")
print(array[:2, :2])

print("Last two rows and last two columns:")
print(array[-2:, -2:])

print("All rows, second column:")
print(array[:, 1])

print("Second row, all columns:")
print(array[1, :])
```

```
4x4 Array with numbers from 1 to 16:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]

Array Slicing Operations:
First two rows and first two columns:
[[1 2]
 [5 6]]
Last two rows and last two columns:
[[11 12]
 [15 16]]
All rows, second column:
[ 2  6 10 14]
Second row, all columns:
[5 6 7 8]
```

```
#17th question
array_2d = np.arange(12).reshape(6, 2)

array_3d = array_2d.reshape(2, 3, 2)

flattened_array = array_3d.flatten()

print("Original 2D array:")
print(array_2d)

print("\nReshaped 3D array:")
print(array_3d)

print("\nFlattened array:")
print(flattened_array)
```

```
Original 2D array:
[[ 0  1]
 [ 2  3]
 [ 4  5]
```

```
 [ 6  7]
 [ 8  9]
 [10 11]]

Reshaped 3D array:
[[[ 0  1]
  [ 2  3]
  [ 4  5]]

 [[ 6  7]
  [ 8  9]
  [10 11]]]

Flattened array:
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

```
#18th question
array_2d = np.array([[1, 2, 3],
                     [4, 5, 6],
                     [7, 8, 9]])

array_1d = np.array([10, 20, 30])

result = array_2d + array_1d

print("Original 3x3 array:")
print(array_2d)

print("\n1x3 array to be broadcasted:")
print(array_1d)

print("\nResult after broadcasting and addition:")
print(result)
```

```
Original 3x3 array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

1x3 array to be broadcasted:
[10 20 30]

Result after broadcasting and addition:
[[11 22 33]
 [14 25 36]
 [17 28 39]]
```

```
#19th question
A = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])
B = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

add_result = A + B
subtract_result = A - B
multiply_result = A * B
divide_result = A / B

print("Element-wise Addition:\n", add_result)
print("\nElement-wise Subtraction:\n", subtract_result)
print("\nElement-wise Multiplication:\n", multiply_result)
print("\nElement-wise Division:\n", divide_result)
```

```
Element-wise Addition:
 [[11 22 33]
 [44 55 66]
 [77 88 99]]

Element-wise Subtraction:
 [[ 9 18 27]
 [36 45 54]
 [63 72 81]]

Element-wise Multiplication:
 [[ 10  40  90]
 [160 250 360]
 [490 640 810]]

Element-wise Division:
 [[10. 10. 10.]
 [10. 10. 10.]
 [10. 10. 10.]]
```

```
#20th question
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Course1': [85, 90, 88],
```

```
        'Course2': [92, 80, 76],
        'Course3': [78, 85, 90]
    }

    df = pd.DataFrame(data)
    df['Total'] = df['Course1'] + df['Course2'] + df['Course3']

    def grade(total):
        if total >= 250:
            return 'A'
        elif total >= 200:
            return 'B'
        else:
            return 'C'

    df['Grade'] = df['Total'].apply(grade)

    print("\nUpdated DataFrame with 'Total' and 'Grade':")
    print(df)
```

```
    Updated DataFrame with 'Total' and 'Grade':
          Name  Course1  Course2  Course3  Total Grade
    0    Alice       85       92       78    255     A
    1      Bob       90       80       85    255     A
    2  Charlie       88       76       90    254     A
```

Start coding or generate with AI.