HW1 ASSIGNMENT

Q 2.1)

1. Write a code for the mean and find the following values:  
   μ(X) = ?

μ(Y ) = ?

μ(Z) = ?

* Code:-

import numpy as np

X = [3, 1, 2, 3, 1, 4, 3, 3]

Y = [1, 4, 3, 1, 1, 3]

Z = [3, 3, 1, 4, 2, 1, 4, 2]

mean\_X = np.mean(X)

mean\_Y = np.mean(Y)

mean\_Z = np.mean(Z)

print("Mean (X):", mean\_X)

print("Mean (Y):", mean\_Y)

print("Mean (Z):", mean\_Z)

1. Write a code for the harmonic mean and find the following values:  
   μh(X) = ?

μh(Y ) = ?

μh(Z) = ?

* Code:-

from scipy.stats import hmean

harmonic\_mean\_X = hmean(X)

harmonic\_mean\_Y = hmean(Y)

harmonic\_mean\_Z = hmean(Z)

print("Harmonic Mean (X):", harmonic\_mean\_X)

print("Harmonic Mean (Y):", harmonic\_mean\_Y)

print("Harmonic Mean (Z):", harmonic\_mean\_Z)

1. Write a code for the geometric mean and find the following values:  
   μg(X) = ?

μg(Y ) = ?

μg(Z) = ?

* Code:-

from scipy.stats import gmean

geometric\_mean\_X = gmean(X)

geometric\_mean\_Y = gmean(Y)

geometric\_mean\_Z = gmean(Z)

print("Geometric Mean (X):", geometric\_mean\_X)

print("Geometric Mean (Y):", geometric\_mean\_Y)

print("Geometric Mean (Z):", geometric\_mean\_Z)

1. Write a code for the arithmetic-geometric mean and find the following values:  
   μag(X) = ?

μag(Y ) = ?

μag(Z) = ?

* Code:-

def arithmetic\_geometric\_mean(a, b, tol=1e-9):

while abs(a - b) > tol:

a, b = (a + b) / 2, np.sqrt(a \* b)

return a

def compute\_agm(data):

a = np.mean(data)

g = gmean(data)

return arithmetic\_geometric\_mean(a, g)

agm\_X = compute\_agm(X)

agm\_Y = compute\_agm(Y)

agm\_Z = compute\_agm(Z)

print("Arithmetic-Geometric Mean (X):", agm\_X)

print("Arithmetic-Geometric Mean (Y):", agm\_Y)

print("Arithmetic-Geometric Mean (Z):", agm\_Z)

1. Write a code for the arithmetic-geometric mean and find the following values:  
   μag(X) = ?

μag(Y ) = ?

μag(Z) = ?

* Code:-

def arithmetic\_harmonic\_geometric\_mean(data, tol=1e-9):

a = np.mean(data) # Arithmetic mean

h = hmean(data) # Harmonic mean

g = gmean(data) # Geometric mean

while abs(a - h) > tol or abs(a - g) > tol:

a, h, g = (a + h + g) / 3, hmean([a, h, g]), gmean([a, h, g])

return a

ahg\_X = arithmetic\_harmonic\_geometric\_mean(X)

ahg\_Y = arithmetic\_harmonic\_geometric\_mean(Y)

ahg\_Z = arithmetic\_harmonic\_geometric\_mean(Z)

print("Arithmetic-Harmonic-Geometric Mean (X):", ahg\_X)

print("Arithmetic-Harmonic-Geometric Mean (Y):", ahg\_Y)

print("Arithmetic-Harmonic-Geometric Mean (Z):", ahg\_Z)

1. Write a code for the median and find the following values:  
   μi(X) = ?

μi(Y ) = ?

μi(Z) = ?

* Code:-

median\_X = np.median(X)

median\_Y = np.median(Y)

median\_Z = np.median(Z)

print("Median (X):", median\_X)

print("Median (Y):", median\_Y)

print("Median (Z):", median\_Z)

1. Write a code for the mode and find the following values:  
   μo(X) = ?

μo(Y ) = ?

μo(Z) = ?

* Code:-

from scipy.stats import mode

mode\_X = mode(X).mode[0]

mode\_Y = mode(Y).mode[0]

mode\_Z = mode(Z).mode[0]

print("Mode (X):", mode\_X)

print("Mode (Y):", mode\_Y)

print("Mode (Z):", mode\_Z)

1. Write a code for the midpoint and find the following values:  
   μm(X) = ?

μm(Y ) = ?

μm(Z) = ?

* Code:-

midpoint\_X = (min(X) + max(X)) / 2

midpoint\_Y = (min(Y) + max(Y)) / 2

midpoint\_Z = (min(Z) + max(Z)) / 2

print("Midpoint (X):", midpoint\_X)

print("Midpoint (Y):", midpoint\_Y)

print("Midpoint (Z):", midpoint\_Z)

1. Write a code for the p% trimmed midpoint and find the following values:  
   μm(X25∼75%) = ?

μm(Y25∼75%) = ?

μm(Z25∼75%) = ?

* Code:-

from scipy.stats import trim\_mean

def trimmed\_midpoint(data, lower=0.25, upper=0.75):

# Sort the data

sorted\_data = sorted(data)

# Find the range (25% to 75%)

lower\_bound = int(len(data) \* lower)

upper\_bound = int(len(data) \* upper)

trimmed\_data = sorted\_data[lower\_bound:upper\_bound]

return (min(trimmed\_data) + max(trimmed\_data)) / 2

trimmed\_midpoint\_X = trimmed\_midpoint(X)

trimmed\_midpoint\_Y = trimmed\_midpoint(Y)

trimmed\_midpoint\_Z = trimmed\_midpoint(Z)

print("Trimmed Midpoint (X 25%~75%):", trimmed\_midpoint\_X)

print("Trimmed Midpoint (Y 25%~75%):", trimmed\_midpoint\_Y)

print("Trimmed Midpoint (Z 25%~75%):", trimmed\_midpoint\_Z)

1. Write a code for the exclusive quartiles and find the following values:  
   xQ1(X) = ?

xQ1(Y ) = ?

xQ1(Z) = ?  
xQ3(X) = ?

xQ3(Y ) = ?

xQ3(Z) = ?

* Code:-

def exclusive\_quartiles(data):

q1 = np.percentile(data, 25, method='midpoint')

q3 = np.percentile(data, 75, method='midpoint')

return q1, q3

q1\_X, q3\_X = exclusive\_quartiles(X)

q1\_Y, q3\_Y = exclusive\_quartiles(Y)

q1\_Z, q3\_Z = exclusive\_quartiles(Z)

print("Exclusive Quartiles (Q1, Q3) for X:", q1\_X, q3\_X)

print("Exclusive Quartiles (Q1, Q3) for Y:", q1\_Y, q3\_Y)

print("Exclusive Quartiles (Q1, Q3) for Z:", q1\_Z, q3\_Z)

1. Write a code for the inclusive quartiles and find the following values:  
   iQ1(X) = ?

iQ1(Y ) = ?

iQ1(Z) = ?  
iQ3(X) = ?

iQ3(Y ) = ?

iQ3(Z) = ?

* Code:-

def inclusive\_quartiles(data):

q1 = np.percentile(data, 25, method='midpoint')

q3 = np.percentile(data, 75, method='midpoint')

return q1, q3

iq1\_X, iq3\_X = inclusive\_quartiles(X)

iq1\_Y, iq3\_Y = inclusive\_quartiles(Y)

iq1\_Z, iq3\_Z = inclusive\_quartiles(Z)

print("Inclusive Quartiles (iQ1, iQ3) for X:", iq1\_X, iq3\_X)

print("Inclusive Quartiles (iQ1, iQ3) for Y:", iq1\_Y, iq3\_Y)

print("Inclusive Quartiles (iQ1, iQ3) for Z:", iq1\_Z, iq3\_Z)

1. Write a code for the entity proportional quartiles and find the following values:  
   eQ1(X) = ?

eQ1(Y ) = ?

eQ1(Z) = ?  
eQ3(X) = ?

eQ3(Y ) = ?

eQ3(Z) = ?

* Code:-

def entity\_proportional\_quartiles(data):

sorted\_data = sorted(data)

n = len(sorted\_data)

q1 = sorted\_data[int(0.25 \* (n + 1)) - 1]

q3 = sorted\_data[int(0.75 \* (n + 1)) - 1]

return q1, q3

eq1\_X, eq3\_X = entity\_proportional\_quartiles(X)

eq1\_Y, eq3\_Y = entity\_proportional\_quartiles(Y)

eq1\_Z, eq3\_Z = entity\_proportional\_quartiles(Z)

print("Entity Proportional Quartiles (eQ1, eQ3) for X:", eq1\_X, eq3\_X)

print("Entity Proportional Quartiles (eQ1, eQ3) for Y:", eq1\_Y, eq3\_Y)

print("Entity Proportional Quartiles (eQ1, eQ3) for Z:", eq1\_Z, eq3\_Z)

1. Write a code for the scale proportional quartiles and find the following values:  
   sQ1(X) = ?

sQ1(Y ) = ?

sQ1(Z) = ?  
sQ3(X) = ?

sQ3(Y ) = ?

sQ3(Z) = ?

* Code:-

def scale\_proportional\_quartiles(data):

sorted\_data = sorted(data)

n = len(sorted\_data)

q1\_index = 0.25 \* (n - 1)

q3\_index = 0.75 \* (n - 1)

# Interpolation for Q1

lower\_q1 = int(np.floor(q1\_index))

upper\_q1 = int(np.ceil(q1\_index))

q1 = (sorted\_data[lower\_q1] + sorted\_data[upper\_q1]) / 2 if lower\_q1 != upper\_q1 else sorted\_data[lower\_q1]

# Interpolation for Q3

lower\_q3 = int(np.floor(q3\_index))

upper\_q3 = int(np.ceil(q3\_index))

q3 = (sorted\_data[lower\_q3] + sorted\_data[upper\_q3]) / 2 if lower\_q3 != upper\_q3 else sorted\_data[lower\_q3]

return q1, q3

sq1\_X, sq3\_X = scale\_proportional\_quartiles(X)

sq1\_Y, sq3\_Y = scale\_proportional\_quartiles(Y)

sq1\_Z, sq3\_Z = scale\_proportional\_quartiles(Z)

print("Scale Proportional Quartiles (sQ1, sQ3) for X:", sq1\_X, sq3\_X)

print("Scale Proportional Quartiles (sQ1, sQ3) for Y:", sq1\_Y, sq3\_Y)

print("Scale Proportional Quartiles (sQ1, sQ3) for Z:", sq1\_Z, sq3\_Z)

Q 2.2)

During the Peloponnesian war in 431 BC, attackers besieging Plataea wanted to find  
the height of the wall in order to build ladders needed. The height of the wall was estimated  
by counting the number of bricks. 40 soldiers reported their estimates of the number of  
bricks. Frequency of soldiers estimated each number of bricks is given as follows:

A table with numbers and a tower

AI-generated content may be incorrect.

**The Data**

The data consists of:

* X: Number of bricks
* fr(X): Frequency of estimates for each number of bricks

We first calculate the actual dataset by "expanding" the frequencies into a list of values.

import numpy as np

from scipy.stats import hmean, gmean, mode

# Frequency distribution

X = np.array([103, 104, 105, 106, 107, 108, 109, 110, 111]) # Brick counts

frequencies = np.array([5, 4, 6, 5, 6, 8, 3, 2, 1]) # Frequencies

# Expand the data

data = np.repeat(X, frequencies)

1. Write a code for the arithmetic mean and find the value, μ(B).

# Arithmetic mean

arithmetic\_mean = np.mean(data)

print("Arithmetic Mean (μ(B)):", arithmetic\_mean)

1. Write a code for the median and find the value, μi(B).

# Median

median = np.median(data)

print("Median (μ\_i(B)):", median)

1. Write a code for the mode and find the value, μo(B).

# Mode

mode\_value = mode(data).mode[0]

print("Mode (μ\_o(B)):", mode\_value)

1. Write a code for the harmonic mean and find the value, μh(B).

# Harmonic mean

harmonic\_mean = hmean(data)

print("Harmonic Mean (μ\_h(B)):", harmonic\_mean)

1. Write a code for the geometric mean and find the value, μg(B).

# Geometric mean

geometric\_mean = gmean(data)

print("Geometric Mean (μ\_g(B)):", geometric\_mean)

1. Write a code for the arithmetic-geometric mean and find the value, μag(B).

A black background with white text

AI-generated content may be incorrect.

def arithmetic\_geometric\_mean(a, g, tol=1e-9):

while abs(a - g) > tol:

a, g = (a + g) / 2, np.sqrt(a \* g)

return a

# Initial values

a0 = arithmetic\_mean

g0 = geometric\_mean

# Compute the arithmetic-geometric mean

ag\_mean = arithmetic\_geometric\_mean(a0, g0)

print("Arithmetic-Geometric Mean (μ\_ag(B)):", ag\_mean)