| Challenge | Challenge |
|---|---|
| Day 4: Geometric Distribution II | Day 8: Least Square Regression Line |
| Day 5: Poisson Distribution I | Day 9: Multiple Linear Regression |

## Day 0: Mean, Median, and Mode

$$mean = \mu = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$n = \text{number of values in data set}$$

$$\begin{matrix} X[\frac{n}{2}] & \text{if n is even} \\ \frac{(X[\frac{n-1}{2}]+X[\frac{n+1}{2}])}{2} & \text{if n is odd} \end{matrix}$$

$$X = \text{ordered list of values in data set}$$

$$mode = \text{a number that appears most frequently in a data set}$$

GitHub

```
N = int(input())
sample = sorted(list(map(float, input().strip().split())))
if N == len(sample):

        # mean
        sum_ = 0
        for i in sample:
                sum_ += i
        mean = sum_/N

        # median
        if N%2 == 0:
                median = (sample[int(N/2-0.5)] + sample[int(N/2+0.5)])/2
        else:
                median = sample[int(N/2+0.5)]

        # mode
        mode = []
        count = {}
        for i in sample:
                if i in count:
                        count[i] += 1
                else:
                        count[i] = 1
        for key in count.keys():
                if count[key] == max(count.values()):
                        mode.append(key)

        print(round(mean, 1))
```

```
        print(round(median, 1))
        print(mode[0])
```

## Day 0: Weighted Mean

$$\text{Weighted Mean} = m_w = \frac{\sum_{i=1}^{n}(x_i \times w_i)}{\sum_{i=1}^{n} w_i}$$

GitHub

```
def weightedMean(X, W):
        global n
        XW_sum = 0
        W_sum = 0
        for i in range(n):
                XW_sum += X[i] * W[i]
                W_sum += W[i]
        print(round((XW_sum/W_sum), 1))

if __name__ == '__main__':
        n = int(input().strip())
        vals = list(map(int, input().rstrip().split()))
        weights = list(map(int, input().rstrip().split()))
        weightedMean(vals, weights)
```

## Day 1: Quartiles

$$\text{Lower Quartile} = Q_1 = median(X_1^{n/2})$$

$$\text{Middle Quartile} = Q_2 = median(X_1^{n})$$

$$\text{Upper Quartile} = Q_3 = median(X_{n/2}^{n})$$

$X = \text{ordered list of values in data set} X_i^j = \text{range of X values in between i and j} whereas; i < j, ; 1 \leq i \leq n, ; 1 \leq j \leq n$

GitHub

```
import os

def medium(arr):
        n = len(arr)
        if n%2 == 0:
                Q = (arr[int((n-1)*0.5)] + arr[int((n+1)*0.5)])/2
        else:
                Q = arr[int(n*0.5)]
```

```
        if Q - int(Q) == 0:
                return int(Q)
        else:
                return Q

def quartiles(arr):
        n = len(arr)
        Q2 = medium(arr)
        Q1 = medium(arr[:n//2])
        Q3 = medium(arr[int(n/2+0.5):])
        return Q1, Q2, Q3

if __name__ == '__main__':
        fptr = open(os.environ['OUTPUT_PATH'], 'w')
        n = int(input().strip())
        data = list(sorted(map(int, input().rstrip().split())))
        res = quartiles(data)
        fptr.write('\n'.join(map(str, res)))
        fptr.write('\n')
        fptr.close()
```

## Day 1: Standard Deviation

$$\text{Standard Deviation} = \sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}$$

GitHub

```
def mean(arr):
        global n
        sum_ = 0
        for i in range(n):
                sum_ += arr[i]
        return n, sum_/n

def stdDev(arr):
        n, mean_ = mean(arr)
        std = 0
        for i in range(n):
                std += ((mean_ - arr[i])**2)
        print((std/n)**0.5)

if __name__ == '__main__':
        n = int(input().strip())
        vals = list(map(int, input().rstrip().split()))
        stdDev(vals)
```

### Day 1: Interquartile Range

$$\text{Interquartile Range} = Q3 - Q1$$

GitHub

```
def median(arr):
        N = len(arr)
        if N%2==0:
                med = (arr[int((N-1)*0.5)] + arr[int((N+1)*0.5)])/2
        else:
                med = arr[int(N*0.5)]
        return float(med)

def interQuartile(values, freqs):
        global n
        value_list = []
        for i in range(n):
                for _ in range(freqs[i]):
                        value_list.append(values[i])
        value_list.sort()
        n = len(value_list)
        print(round(\
        median(value_list[int((n+1)*0.5):]) - median(value_list[:n//2]), 1))

if __name__ == '__main__':
        n = int(input().strip())
        val = list(map(int, input().rstrip().split()))
        freq = list(map(int, input().rstrip().split()))
        interQuartile(val, freq)
```

### Day 4: Binomial Distribution I

$$b(x, n, p) = \frac{n!}{x!(n-x)!} \cdot p^x \cdot q^{(n-x)}$$

$$b(x \geq r, n, p) = \sum_{i=r}^{n} b(x = i, n, p)$$

GitHub

```
p1, n = map(float, input().split())

# p for boys
gap = 1/(1 + (n/p1))
x = 3
n = 6

def fact(x):
        a = 1
        for i in range(1, x+1):
```

```
                a *= i
        return a

def comb(n, r):
        return float(fact(n)/(fact(r)*fact(n-r)))

def bino(x, n, p):
        q = 1 - p
        return (comb(n, x)*(p**x)*(q**(n-x)))

print(round(sum([bino(i, n, gap) for i in range(x, n+1)]), 3))
```

## Day 4: Binomial Distribution II

$$P(i < x \leq j) = b(i \leq x \leq j, n, p) = \sum_{i=r}^{j} b(x = i, n, p)$$

GitHub

```
p, n = map(int, input().split())

p /= 100

def fact(x):
    if x == 0:
        a = 1
    else:
        a = 1
        for i in range(1, x+1):
            a *= i
    return a

def comb(n, r):
    return float(fact(n)/(fact(r)*fact(n-r)))

def bino(x, n, p):
    q = 1 - p
    return (comb(n, x)*(p**x)*(q**(n-x)))

def prob(from_, to_):
    print(round(sum([bino(i, n, p) for i in range(from_, to_+1)]), 3))

# no more than 2 rejects
prob(0, 2)

# at least 2 rejects
prob(2, n)
```

### Day 4: Geometric Distribution I

$$g(n, p) = q^{n-1} \cdot p$$

$$whereas, ; q = 1 - p$$

GitHub

```
p1, p2 = map(int, input().split())
p0 = int(input())

p = p1/p2

def geo(n, p):
    return round((p*((1-p)**(n-1))), 3)

print(geo(p0, p))
```

### Day 4: Geometric Distribution II

$$P(x \leq j) = g(n \leq j, p) = \sum_{i=1}^{j} g(n = i, p)$$

GitHub

```
p1, p2 = map(int, input().split())
p0 = int(input())

p = p1/p2

def geo(n, p):
    return round(sum([(p*((1-p)**(i-1))) for i in range(1, n+1)]), 3)

print(geo(p0, p))
```

### Day 5: Poisson Distribution I

$$P(k, \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

GitHub

```
from math import exp

mean = float(input())
X = int(input())
```

```
def fact(x):
    if x == 0:
        n = 1
    else:
        n = 1
        for i in range(1, x+1):
            n *= i
    return n

def pois(X, mean):
    return round(((mean**X)*(exp(-mean)))/fact(X), 3)

print(pois(X, mean))
```

## Day 5: Poisson Distribution II

$$E[X^2] = \lambda + \lambda^2$$

GitHub

```
X, Y = map(float, input().split())

#def C_A(X):
print(round(160 + (40*(X+X**2)), 3))

#def C_B(Y):
print(round(128 + (40*(Y+Y**2)), 3))
```

## Day 5: Normal Distribution I

$$P(X \leq x) = F_X(x) = \frac{1}{2}\Big(1 + erf\Big(\frac{x-\mu}{\sigma\sqrt{2}}\Big)\Big)$$

$$P(a \leq X \leq b) = F_X(b) - F_X(a)$$

GitHub

```
from math import exp, sqrt, pi, erf

# read input
mean, std = map(float, input().split())
X1 = float(input())
X2, X3 = map(float, input().split())

'''
# pdf: normal distribution
```

```
def norm(x, mean, std):
    con1 = exp(-((x - mean)**2)/(2*(std**2)))
    return con1/(std*sqrt(2*pi))
'''

# cdf: normal distribution
def norm(x, mean, std):
    return ((1 + erf((x - mean)/(sqrt(2)*std)))*0.5)

# Q01
print(round(norm(X1, mean, std), 3))

# Q02
print(round((norm(X3, mean, std) - norm(X2, mean, std)), 3))
```

## Day 5: Normal Distribution II

$$P(X \leq x) = F_X(x)$$

$$P(X > x) = 1 - P(X \leq x) = 1 - F_X(x)$$

GitHub

```
from math import sqrt, erf

mu, std = map(float, input().split())
X1 = float(input())
X2 = float(input())

# cdf: normal distribution
def norm(x, mean, std):
    return ((1 + erf((x - mean)/(sqrt(2)*std)))*0.5)*100

# X > X1
print(round(100 - norm(X1, mu, std), 2))

# X >= X2
print(round(100 - norm(X2, mu, std), 2))

# X < X2
print(round(norm(X2, mu, std), 2))
```

## Day 6: The Central Limit Theorem I

$$\mu' = n \times \mu$$

$$\sigma' = \sqrt{n} \times \sigma$$

$$F_X(x, \mu', \sigma') = \frac{1}{2}\Big(1 + erf\Big(\frac{x - \mu'}{\sigma'\sqrt{2}}\Big)\Big)$$

GitHub

```
from math import sqrt, erf

# read inputs
max_ = int(input())
box = int(input())
mu = int(input())
std = int(input())

# cdf: normal distribution
def norm(x, mean, std):
    return ((1 + erf((x - mean)/(sqrt(2)*std)))*0.5)

mu_ = box*mu
std_ = sqrt(box)*std

print(round(norm(max_, mu_, std_), 4))
```

## Day 6: The Central Limit Theorem II

$$\mu' = n \times \mu$$

$$\sigma' = \sqrt{n} \times \sigma$$

$$F_X(x, \mu', \sigma') = \frac{1}{2}\Big(1 + erf\Big(\frac{x - \mu'}{\sigma'\sqrt{2}}\Big)\Big)$$

GitHub

```
from math import sqrt, erf

# read all inputs
max_tic = int(input())
tic = int(input())
mu = float(input())
std = float(input())

# CDF: normal distribution
def norm(x, mean, std):
    return ((1 + erf((x - mean)/(sqrt(2)*std)))*0.5)

mu_ = tic * mu
std_ = sqrt(tic) * std

print(round(norm(max_tic, mu_, std_), 4))
```

## Day 6: The Central Limit Theorem III

$$\text{Confidence Interval} = \bar{x} \pm z\frac{\sigma}{\sqrt{n}}$$

GitHub

```
# read inputs
sample = int(input())
mu = int(input())
std = int(input())
inte = float(input())
z = float(input())

# lower limit
print(round(mu - z*(std/(sample)**0.5), 2))

# higher limit
print(round(mu + z*(std/(sample)**0.5), 2))
```

## Day 7: Pearson Correlation Coefficient I

$$\mu_X = \frac{\sum_n^{} {}_{i=1} x_i}{n}, similarly; \mu_Y$$

$$\sigma_X = \sqrt{\frac{\sum_n^{} {}_{i=1}(x_i - \mu)^2}{n}}, similarly; \sigma_Y$$

$$\rho_{X,Y} = \frac{\Sigma(x_i - \mu_X) \cdot (y_i - \mu_Y)}{n \cdot \sigma_X \cdot \sigma_Y}$$

GitHub

```
# Read inputs
n = int(input())
X = list(map(float, input().split()))
Y = list(map(float, input().split()))

# check n == len(X) == len(Y)

# mean
def mean(a):
    return sum(a)/len(a)

# standard deviation
def std(a):
    global n
```

```
    mu = mean(a)
    sum_ = 0.0
    for i in range(n):
        sum_ += ((a[i] - mu)**2)
    return (sum_/n)**0.5

# corr(X, Y)
def corr(arr1, arr2):
    global n
    mean1 = mean(arr1)
    mean2 = mean(arr2)
    conv = 0.0
    for i in range(n):
        conv += (arr1[i] - mean1) * (arr2[i] - mean2)
    print(round(conv/(n*std(arr1)*std(arr2)), 3))

corr(X, Y)
```

## Day 7: Spearman's Rank Correlation Coefficient

$$r_x\text{: rank of X values in descending order}$$

$$r_y\text{: rank of Y values in descending order}$$

$$N\text{: number of values in X or Y, } N_X = N_Y$$

$$1 \leq r_x, r_y \leq N$$

$$r_{xy} = 1 - \frac{6\Sigma(r_x - r_y)^2}{N(N^2 - 1)}$$

GitHub

```
# Read inputs
n = int(input())
X = list(map(float, input().split()))
Y = list(map(float, input().split()))

# Ranking
def rank(arr):
    return [sorted(arr).index(x)+1 for x in arr]

# Spearman's rank correlation
def corr(arr1, arr2):
    global n
    r_arr1 = rank(arr1)
    r_arr2 = rank(arr2)
    sum_ = 0.0
    for i in range(n):
        sum_ += (r_arr1[i] - r_arr2[i])**2
    print(round(1-((6*sum_)/(n*((n**2)-1))), 3))

corr(X, Y)
```

## Day 8: Least Square Regression Line

$$a = \frac{n\Sigma(x_i y_j) - (\Sigma x_i)(\Sigma y_j)}{n\Sigma(x_i^2) - \Sigma(x_i)^2}$$

$$b = \frac{\Sigma(x_i - \mu_X) \cdot (y_i - \mu_Y)}{n \cdot \sigma_X^2}$$

$$\hat{Y} = a + bX$$

GitHub

```
# Read inputs into array
n = 5
X, Y = list(), list()
for _ in range(n):
    x, y = map(int, input().split())
    X.append(x)
    Y.append(y)

# mean
def mean(a):
    return sum(a)/len(a)

# standard deviation
def std(a):
    global n
    mu = mean(a)
    sum_ = 0.0
    for i in range(n):
        sum_ += ((a[i] - mu)**2)
    return (sum_/n)**0.5

# corr(X, Y)
def corr(arr1, arr2):
    global n
    mean1 = mean(arr1)
    mean2 = mean(arr2)
    conv = 0.0
    for i in range(n):
        conv += (arr1[i] - mean1) * (arr2[i] - mean2)
    return conv/(n*std(arr1)*std(arr2))

# fit the best-fit line using least squares and find the value of y
def liner(X, Y, x1):
    b = corr(X,Y)*(std(Y)/std(X))
    a = mean(Y) - (b*mean(X))
    print(round(a + (b*x1), 3))

liner(X, Y, 80)
```

## Day 9: Multiple Linear Regression

GitHub

*Working on a bug. Will be posted soon.*

If you encounter any error, feel free to post your issue in GitHub.



### Releases

No releases published

### Packages

No packages published

### Environments 1

github-pages Active

### Languages

- Python 100.0%