In [5]:
```python
#1a. Write a program to enter name and display as
# “Hello, Name”.

name=input("Enter your name: ")
name=name.capitalize()
print(f"Hello, {name}")
```

```
Hello, Vasudev
```

In [9]:
```python
#1b. WAP to create hashtag after taking two input strings from user.
first=input("Enter first string: ")
second=input("Enter second string: ")
hashtag=first[:3].upper()+second[-3:].upper()
print(f"Your system generated Hashtag is: {hashtag}.")
```

```
Your system generated Hashtag is: SOAMYA.
```

In [18]:
```python
# 2. WAP to compute the roots of a quadratic equation.
a=int(input("Enter the coefficient of x\N{superscript two}: "))
b=int(input("Enter the coefficient of x: "))
c=int(input("Enter the constant: "))
d=b**2 -(4*a*c)
if d==0:
    x=(-b/(2*a))
    print(f"Equation has two real and equal roots: {x:.2f}")
elif d>0:
    x1=((-b+(d**0.5))/(2*a))
    x2=((-b-(d**0.5))/(2*a))
    print(f"Equation has two real and unequal roots: {x1:.2f} and {x2:.2f}")
else:
    img=(((4*a*c)-(b**2))**0.5)/(2*a)
    print(f"Equation has conjugate complex roots: {-b/(2*a):.2f} + {img:.2f} i and {-b/(2*a):.2f} - {img:.2f} i.")
```

```
Equation has conjugate complex roots: -1.50 + 1.32 i and -1.50 - 1.32 i.
```

In [28]:
```python
# 3. WAP to play Stone, Paper, Scissors with Computer.
import random
user=input("Type your option(from 'stone','paper','scissor'): ")
machine=random.choice(['stone','paper','scissor'])
```

```
print(f"Machine chose {machine}.")
print(f"User chose {user}.")
if(user=='stone'):
    if(machine==user):
        print("Draws\N{grinning face}")
    elif(machine=='scissor'):
        print("User Wins\N{grinning face with smiling eyes}")
    else:
        print("Machine Wins\N{disappointed face}")
elif(user=='scissor'):
    if(machine==user):
        print("Draws\N{grinning face}")
    elif(machine=='paper'):
        print("User Wins\N{grinning face with smiling eyes}")
    else:
        print("Machine Wins\N{disappointed face}")
elif(user=="paper"):
    if(machine==user):
        print("Draws\N{grinning face}")
    elif(machine=='scissor'):
        print("Machine Wins\N{disappointed face}")
    else:
        print("User Wins\N{grinning face with smiling eyes}")
```

```
Machine chose scissor.
User chose stone.
User Wins😁
```

In [32]:

```
# 4. Write a program for BMI Calculator with Categorization of underweight, normal weight and overweight.
w=float(input("Enter your Weight in Kg: "))
h=float(input("Enter your Height in metre: "))
bmi=w/(h**2)
print(f"Your BMI score is {bmi:.2f}. ")
if bmi<=18.4:
    print("You are Underweight.")
elif 18.4<bmi and bmi<=25:
    print("You are Healthy.")
else:
    print("You are Overweight.")
```

```
Your BMI score is 26.23.
You are Overweight.
```

In [36]:
```python
# 5. WAP to demonstrate exception handling of Zero Division Error.
a=int(input("Enter Numerator: "))
b=int(input("Enter Denominator: "))
try:
    division=(a/b)
except ZeroDivisionError:
    print("Denominator must be Non-Negative.")
    print("Run Again")
else:
    print(f"Result: {division:.2f}")
finally:
    print("Thank You")
```

```
Result: 2.00
Thank You
```

In [41]:
```python
# 6. WAP to demonstrate OOPs using user defined Cuboid Class.
class cuboid:
    def __init__(self,length,breadth,height):
        self.l=length
        self.b=breadth
        self.h=height
    def volume(self):
        self.v=(self.l*self.b*self.h)
        print(f"Volume of Cuboid is {self.v}unit\N{superscript three}")
    def display(self):
        print(f"Lenght: {self.l}unit\nBreadth: {self.b}unit\nHeight: {self.h}unit")
c=cuboid(10,10,10)
c.display()
c.volume()
```

```
Lenght: 10unit
Breadth: 10unit
Height: 10unit
Volume of Cuboid is 1000unit³
```

In [59]:
```python
# 7. WAP to demonstrate inheritance in OOPs using user defined Rectangle and Cuboid Class.
class rectangle:
```

```
    def __init__(self,length,breadth):
        self.l=length
        self.b=breadth
    def area(self):
        self.a=self.l*self.b
        return self.a
    def display(self):
        print(f"Lenght: {self.l}unit\nBreadth: {self.b}unit")
class cuboid(rectangle):
    def __init__(self,length,breadth,height):
        super().__init__(length,breadth)
        self.h=height
    def volume(self):
        self.v=super().area()*self.h
        return self.v
    def display(self):
        print(f"Lenght: {self.l}unit\nBreadth: {self.b}unit\nHeight: {self.h}unit")
r=rectangle(2,2)
print("Area of Rectangle is: ",r.area(),"unit\N{superscript two}")
print("_"*50)
c=cuboid(3,2,3)
c.display()
print("Volume of Cuboid is: ",c.volume(),"unit\N{superscript three}")
```

```
Area of Rectangle is:  4 unit²
__________________________________________________
Lenght: 3unit
Breadth: 2unit
Height: 3unit
Volume of Cuboid is:  18 unit³
```

In [62]:

```
# 8. Write a Program to implement Inheritance. Create a class Employee inherit two classes Manager from Employee
#and Clerk from Manager.
class employee:
    def __init__(self,name,eid):
        self.name=name
        self.eid=eid
    def display(self):
        print(f"Name of the Employee: {self.name} and Employee ID is {self.eid}.")
class manager(employee):
    def __init__(self, name, eid,department):
```

```
        super().__init__(name,eid)
        self.department=department
    def display(self):
        super().display()
        print("Department Name: ",self.department)
class clerk(manager):
    def __init__(self, name, eid,department,floor):
        super().__init__(name, eid,department)
        self.floor=floor
    def display(self):
        super().display()
        print("Floor Number: ",self.floor)
c=clerk("A","101","Data","2")
c.display()
```

```
Name of the Employee: A and Employee ID is 101.
Department Name:  Data
Floor Number:  2
```

In [1]:
```
# 9. WAP to demonstrate the demarcation of class variable and instance variable in OOP using Employee Class.
class Employee:
    id=100
    t_count=0
    def __init__(self,name,designation,salary):
        self.eid=Employee.id+1
        self.name=name
        self.designation=designation
        self.salary=salary
        Employee.id+=1
        Employee.t_count+=1
    def display(self):
        print("Employee ID: ",self.eid)
        print("Employee Name: ",self.name)
        print("Employee Designation: ",self.designation)
        print("Employee Salary: ",self.salary)

    # @staticmethod
    def count():
        print("No. of Employees :",Employee.t_count)
e=Employee("A","DA",10)
```

```
e.display()
Employee.count()
```

```
Employee ID:  101
Employee Name:  A
Employee Designation:  DA
Employee Salary:  10
No. of Employees : 1
```

In [3]:

```
# 10. Write a Program to determine EOQ using various inventory models.
class EOQ:
    def __init__(self,lamda,ordercost,inventorycost,production=0,shortagecost=0):
        self.lamda=lamda
        self.ordercost=ordercost
        self.inventorycost=inventorycost
        self.production=production
        self.shortagecost=shortagecost
    def EOQ1(self):
        self.q1=(2*self.ordercost*self.lamda/self.inventorycost)**0.5
        print(f"EOQ1: {self.q1:.2f}")

    def EOQ2(self):
        self.q2=((2*self.ordercost*self.lamda/self.inventorycost)**0.5)
                 *((self.production/(self.production-self.lamda))**0.5)
        print(f"EOQ2: {self.q2:.2f}")

    def EOQ3(self):
        self.q3=((2*self.ordercost*self.lamda/self.inventorycost)**0.5)
                *(((self.shortagecost+self.inventorycost)/self.shortagecost)**0.5)
        print(f"EOQ3: {self.q3:.2f}")

    def EOQ4(self):
        self.q4=((2*self.ordercost*self.lamda/self.inventorycost)**0.5)
                *((self.production/(self.production-self.lamda))**0.5)
                *(((self.shortagecost+self.inventorycost)/self.shortagecost)**0.5)
        print(f"EOQ4: {self.q4:.2f}")

e= EOQ(lamda=10000,ordercost=4,inventorycost=2,production=12000,shortagecost=2)

e.EOQ1()
e.EOQ2()
```

```
e.EOQ3()
e.EOQ4()
```

```
EOQ1: 200.00
EOQ2: 489.90
EOQ3: 282.84
EOQ4: 692.82
```

In [1]:

```
# 11. Write a Program to determine different characteristics using various Queuing models.
import math
class queuing_models:
    def __init__(self,lamda,mu,servers=1,systemcapacity=float('inf')):
        self.lamda=lamda
        self.mu=mu
        self.c=servers
        self.rho=self.lamda/self.mu
        self.k=systemcapacity
    def mm1(self):
        self.L=self.rho/(1-self.rho)
        self.Lq=self.L-self.rho
        self.W=self.L/self.lamda
        self.Wq=self.Lq/self.lamda
        self.Lq_dash=1/(1-self.rho)
        print(f"Expected No. Of Customers in the system: {self.L:.2f}")
        print(f"Expected No. Of Customers in the queue: {self.Lq:.2f}")
        print(f"Expected Time Spent in the system: {self.W:.2f}")
        print(f"Expected Time Spent in the queue: {self.Wq:.2f}")
        print(f"Expected No. of Person waiting when there is atleast one person is waiting: {self.Lq_dash:.2f}")
    def mm1k(self):
        if self.rho==1:
            self.Pn=1/(self.k+1)
            self.L=self.k/2
            self.Lq=(self.k*(self.k-1))/(2*(self.k+1))
            self.lamda_eff=self.lamda*(1-self.Pn)
            self.W=self.L/self.lamda_eff
            self.Wq=self.Lq/self.lamda_eff
            print(f"Expected No. Of Customers in the system: {self.L:.2f}")
            print(f"Expected No. Of Customers in the queue: {self.Lq:.2f}")
            print(f"Expected Time Spent in the system: {self.W:.2f}")
            print(f"Expected Time Spent in the queue: {self.Wq:.2f}")
        else:
```

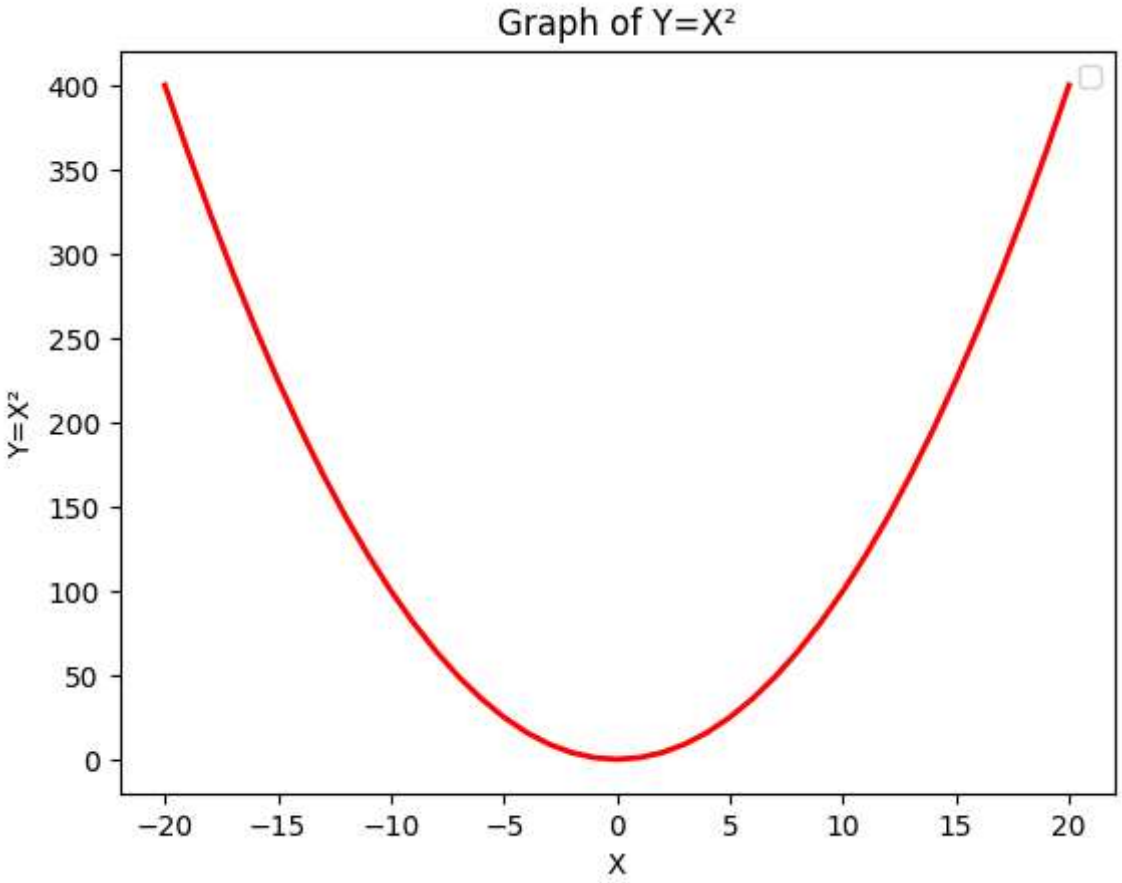
```
            self.Pn=((1-self.rho)/(1-(self.rho)**(self.k+1)))*self.rho**self.k
            self.Lq=(self.rho/(1-self.rho))-(self.rho*((self.k*(self.rho**self.k))+1))/(1-(self.rho**(self.k+1)))
            self.lamda_eff=self.lamda*(1-self.Pn)
            self.L=self.Lq+(self.lamda_eff/self.mu)
            self.W=self.L/self.lamda_eff
            self.Wq=self.Lq/self.lamda_eff
            print(f"Expected No. Of Customers in the system: {self.L:.2f}")
            print(f"Expected No. Of Customers in the queue: {self.Lq:.2f}")
            print(f"Expected Time Spent in the system: {self.W:.2f}")
            print(f"Expected Time Spent in the queue: {self.Wq:.2f}")
    def mmc(self):
        self.r=self.lamda/self.mu
        self.rho=self.lamda/(self.c*self.mu)
        self.Po=((self.r**self.c)/(math.factorial(self.c)*(1-self.rho)) +
                 sum((math.pow(self.r,n))/math.factorial(n) for n in range(self.c)))**(-1)
        self.Lq=math.pow(self.r,self.c)*self.Po*self.rho/(math.factorial(self.c)*math.pow((1-self.rho),2))
        self.L=self.Lq+self.r
        self.W=self.L/self.lamda
        self.Wq=self.Lq/self.lamda
        print(f"Expected No. Of Customers in the system: {self.L:.2f}")
        print(f"Expected No. Of Customers in the queue: {self.Lq:.2f}")
        print(f"Expected Time Spent in the system: {self.W:.2f}")
        print(f"Expected Time Spent in the queue: {self.Wq:.2f}")
q1=queuing_models(8,12)
print("MM1")
q1.mm1()
print("_"*50)
print("MM1K")
q2=queuing_models(6,3,1,5)
q2.mm1k()
print("_"*50)
print("MMC")
q3=queuing_models(6,3,3)
q3.mmc()
```

```
MM1
Expected No. Of Customers in the system: 2.00
Expected No. Of Customers in the queue: 1.33
Expected Time Spent in the system: 0.25
Expected Time Spent in the queue: 0.17
Expected No. of Person waiting when there is atleast one person is waiting: 3.00

--------------------------------------------------
MM1K
Expected No. Of Customers in the system: 4.10
Expected No. Of Customers in the queue: 3.11
Expected Time Spent in the system: 1.39
Expected Time Spent in the queue: 1.05

--------------------------------------------------
MMC
Expected No. Of Customers in the system: 2.89
Expected No. Of Customers in the queue: 0.89
Expected Time Spent in the system: 0.48
Expected Time Spent in the queue: 0.15
```

In [2]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

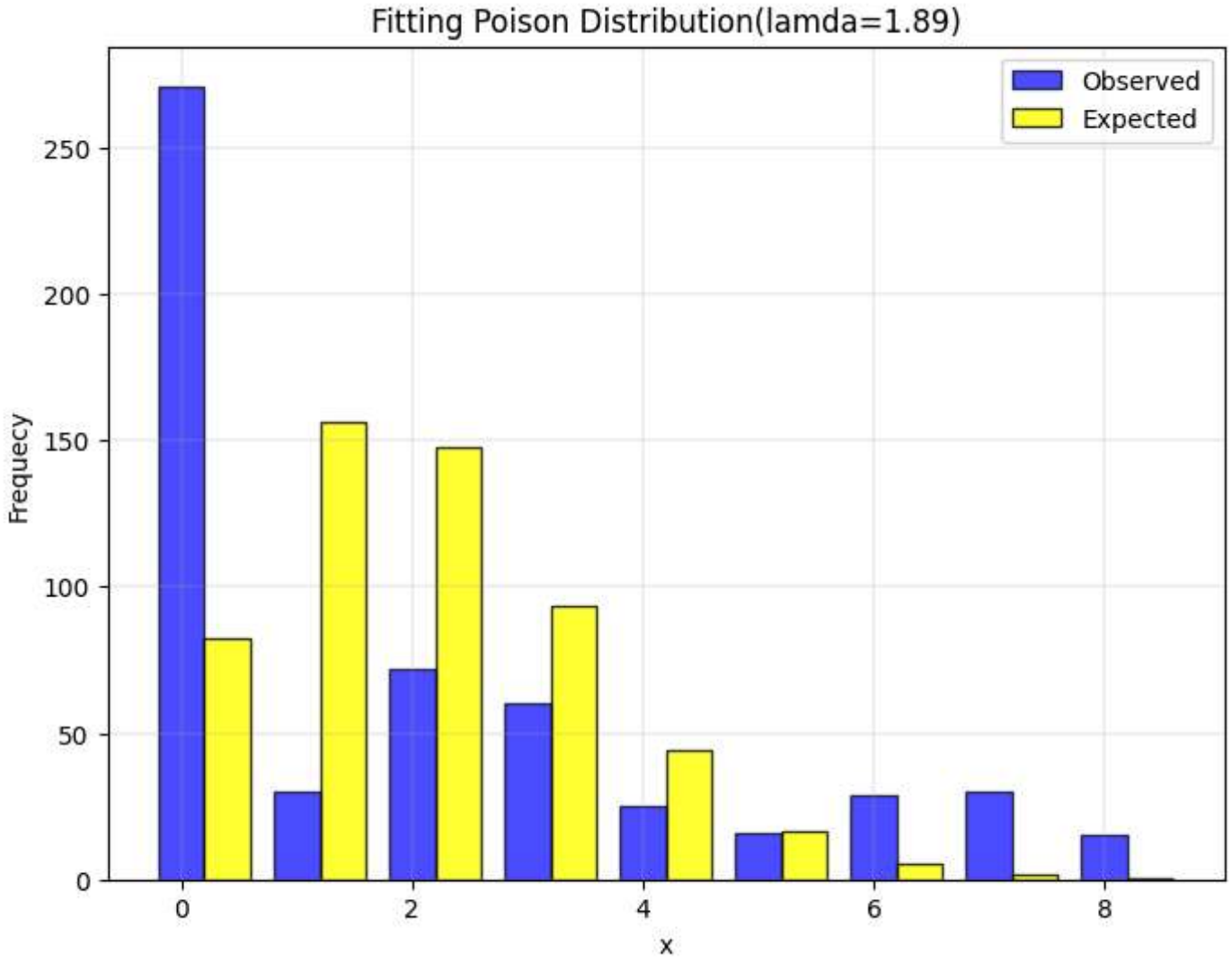
In [30]:
```python
# 12.WAP to plot a graph for function (y=X2).
x=np.arange(-20,21)
y=np.power(x,2)
plt.plot(x,y,color="red",lw=2)
plt.xlabel("X")
plt.ylabel("Y=X\N{superscript two}")
plt.legend()
plt.title("Graph of Y=X\N{superscript two}")
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend
() is called with no argument.
```



In [ ]:
```
#13.WAP to fit poisson distribution on a given data.
# x       0   1   2   3   4   5   6   7   8
# y     271  30  72  60  25  16  29  30  15

from scipy.stats import poisson
x=np.array([0,1,2,3,4,5,6,7,8])
y=np.array([271,30,72,60,25,16,29,30,15])
lamda=np.sum(x*y)/y.sum()
probability=poisson.pmf(x,mu=lamda)
y_expected=y.sum()*probability

plt.figure(figsize=(8,6))
plt.bar(x,y,width=0.4,label="Observed",color="blue",alpha=0.7,edgecolor="black")
```

```
plt.bar(x+0.4,y_expected,width=0.4,label="Expected",color="yellow",alpha=0.8,edgecolor="black")
plt.xlabel("x")
plt.ylabel("Frequecy")
plt.legend()
plt.title(f"Fitting Poison Distribution(lamda={np.round(lamda,2)})")
plt.grid(True,'both',lw=0.2)
plt.show()
```



In [57]:
```
# 14.Write a python function that calculates the pearson correlation cofficient between two lists of numbers.

# Step 1 ] Taking Input from the user
x=[]
n_x=int(input("Enter no. of elements in x"))
for i in range(0,n_x):
    ele=int(input("Enter element"))
    x.append(ele)


y=[]
n_y=int(input("Enter no. of elements in y"))
for i in range(0,n_y):
    ele=int(input("Enter element"))
    y.append(ele)


# Step 2] Defining Formula

def pearson_corr_coeff(x,y):
    x_mean=np.sum(x)/len(x)
    y_mean=np.sum(y)/len(y)
    numerator=np.sum((x-x_mean)*(y-y_mean))
    denominator=((np.sum((x-x_mean)**2))*(np.sum((y-y_mean)**2)))**0.5
    result=numerator/denominator
    return result

print("List 1: ",x)
print("List 2: ",y)

# Step 3] Calling Function
print("User Defined Function: ",pearson_corr_coeff(x,y))

# Step 4] Using Inbuilt function
print("InBuilt Function 1:    ",np.corrcoef(x,y)[0,1])

from scipy.stats import pearsonr
print("InBuilt Function 2:    ",pearsonr(x,y)[0])
```

```
List 1:  [1, 2, 3, 4, 5]
List 2:  [3, 6, 7, 9, 11]
User Defined Function:  0.9904434667711052
InBuilt Function 1:     0.9904434667711051
InBuilt Function 2:     0.9904434667711052
```

In [11]:
```python
# 15. Write a python function that calculates Spearman correlation cofficient.
df=pd.DataFrame({"x":[12,21,3,14,25],"y":[10,20,30,40,50]})
df["rank_x"]=df["x"].rank(axis=0,method="dense")
df["rank_y"]=df["y"].rank(axis=0,method="dense")
df["difference_square"]=(df["rank_x"]-df["rank_y"])**2
n=df["x"].size
numerator=df["difference_square"].sum()
denominator=n*((n**2)-1)
spearman_corr_coeff=1-(numerator/denominator)
print("Spearman Correlation Coefficient: ",spearman_corr_coeff)
df
```

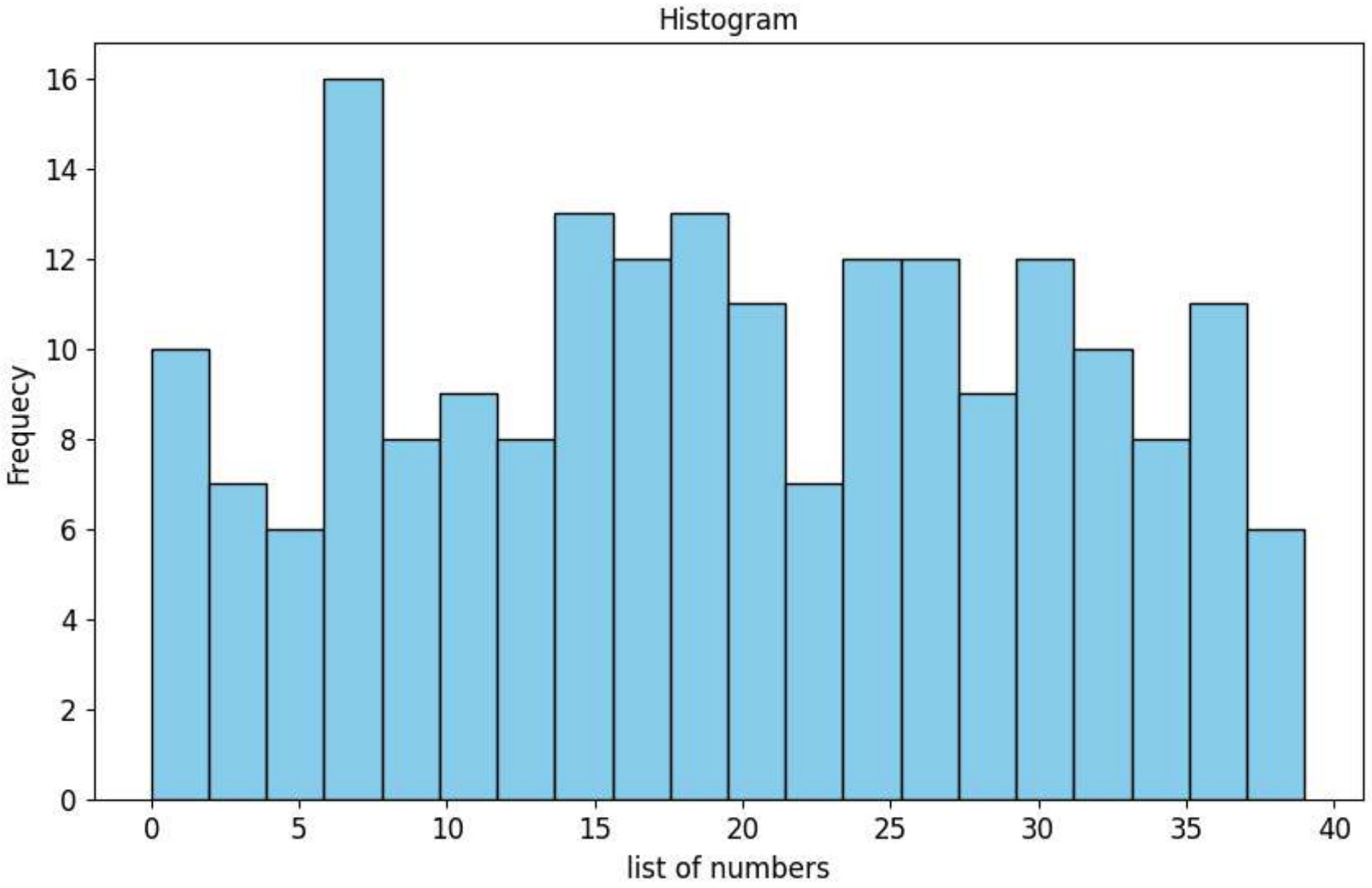
```
Spearman Correlation Coefficient:  0.9166666666666666
```

Out[11]:

	x	y	rank_x	rank_y	difference_square
0	12	10	2.0	1.0	1.0
1	21	20	4.0	2.0	4.0
2	3	30	1.0	3.0	4.0
3	14	40	3.0	4.0	1.0
4	25	50	5.0	5.0	0.0

In [43]:
```python
# 16. Using matplotlib plot histogram of list of numbers.
np.random.seed(40)
list1=np.random.randint(0,40,200)
plt.figure(figsize=(10,6))
plt.hist(list1,bins=20,edgecolor="black",color="skyblue",alpha=1)
plt.xlabel("list of numbers",fontsize=12)
plt.ylabel("Frequecy",fontsize=12)
plt.title("Histogram",fontsize=12)
```

```
plt.tick_params(axis="both",which="major",labelsize=12,color="black")
plt.show()
```



In [60]:
```
# 17. Write a python function that calculates the Z-score for the lists of numbers.

def z_score(x):
    mu=np.mean(x)
```

```
    sigma=np.std(x)
    z=(x-mu)/sigma
    return list(np.round(z,3))


x=[]
n=int(input("Enter the number of elements in list: "))
for i in range(0,n):
    ele=int(input("Enter element: "))
    x.append(ele)


print("List of Numbers: ",x)
print("Z-Score: ",z_score(x))
```

```
List of Numbers:  [1, 2, 3, 4, 5]
Z-Score:  [-1.414, -0.707, 0.0, 0.707, 1.414]
```

In [ ]:

```
# 18. WAP to test the significance of two sample means.

#                Ho : There is no significance difference between two sample means.
#                             Mean of Sample 1 = Mean of Sample 2
#                H1 : There is significance difference between two sample means.
#                             Mean of Sample 1 != Mean of Sample 2
#      Checking at 5% level of significance.


from scipy.stats import t

print("_____Using User-defined Function_____")

def two_sample_ttest(sample1,sample2):
    n1=len(sample1)
    n2=len(sample2)

    xbar1=np.mean(sample1)
    xbar2=np.mean(sample2)

    v1=np.var(sample1,ddof=1)
    v2=np.var(sample2,ddof=1)
```

```
    pooled_var=((n1-1)*v1+(n2-1)*v2)/(n1+n2-2)
    pooled_std=pooled_var**0.5

    ts=(xbar1-xbar2)/(pooled_std*np.sqrt((1/n1)+(1/n2)))

    p_val=t.cdf(ts,n1+n2-2)
    left_wall=t.ppf(0.025,n1+n1-2)
    right_wall=t.ppf(0.975,n1+n1-2)

    print("test statistic: ",round(ts,3))
    print("p-value: ",round(p_val,3))

    if left_wall<ts and ts<right_wall:
        print("We do not have sufficient reason to reject Ho because of test statistic.")
    else:
        print("We have sufficient reason to reject Ho because of test statistic.")

    if p_val<0.025 or p_val>97.5:
        print("We have sufficient reason to reject Ho because of p-value.")
    else:
        print("We do not have sufficient reason to reject Ho because of p-value.")

    return



sample1=[27,32.2,30.4,28,26.5,25.5,29.6,27.2]
sample2=[31.4,29.9,33.2,34.4,32,28.7,26.1,30.3]

two_sample_ttest(sample1,sample2)


print("_____________________________________Using Inbuilt function_____________________________________")

from scipy.stats import ttest_ind

ts,p_val=ttest_ind(sample1,sample2)
print("test statistic: ",round(ts,3))
print("p-value: ",round(p_val/2,3))
```

```
_____________________________________Using User-defined Function_____________________________________
test statistic:  -2.007
p-value:  0.032
We do not have sufficient reason to reject Ho because of test statistic.
We do not have sufficient reason to reject Ho because of p-value.
_____________________________________Using Inbuilt function_____________________________________
test statistic:  -2.007
p-value:  0.032
```

19. WAP to test the goodness of fit of a given dataset on binomial distribution.

A Survey of 800 families with 4 children revealed the following distribution.

X: No. of Boys: 0 1 2 3 4

frequency: 32 178 290 236 64

Is the result consistent with the hypothesis that Male & Female births are equally probable at 95% confidence level.

- Here we have to check for the hypothesis that X ~ Binomial distribution or not with probability=1/2.

In [ ]:
```python
from scipy.stats import chisquare,binom,chi2

x=np.array([0,1,2,3,4])
observed=np.array([32,178,290,236,64])

prob=binom.pmf(x,4,0.5)

expected=prob*800

ts=np.sum(((observed-expected)**2)/expected)

ls=chi2.ppf(0.95,4)
p_val=chi2.sf(ts,4)

print("test statistic: ",round(ts,3))
print("Chi value at 5% level of significance: ",round(ls,3))
print("p-value: ",round(p_val,3))

if ls<ts:
    print("X do not follows Binomial Distribution.")
```

```
else:
    print("X follows Binomial Distribution.")


print("____________________Using inbuilt function____________________")
t_stat,p_val=chisquare(observed,expected)
print("test statistic: ",round(t_stat,3))
print("p-value: ",round(p_val,3))
```

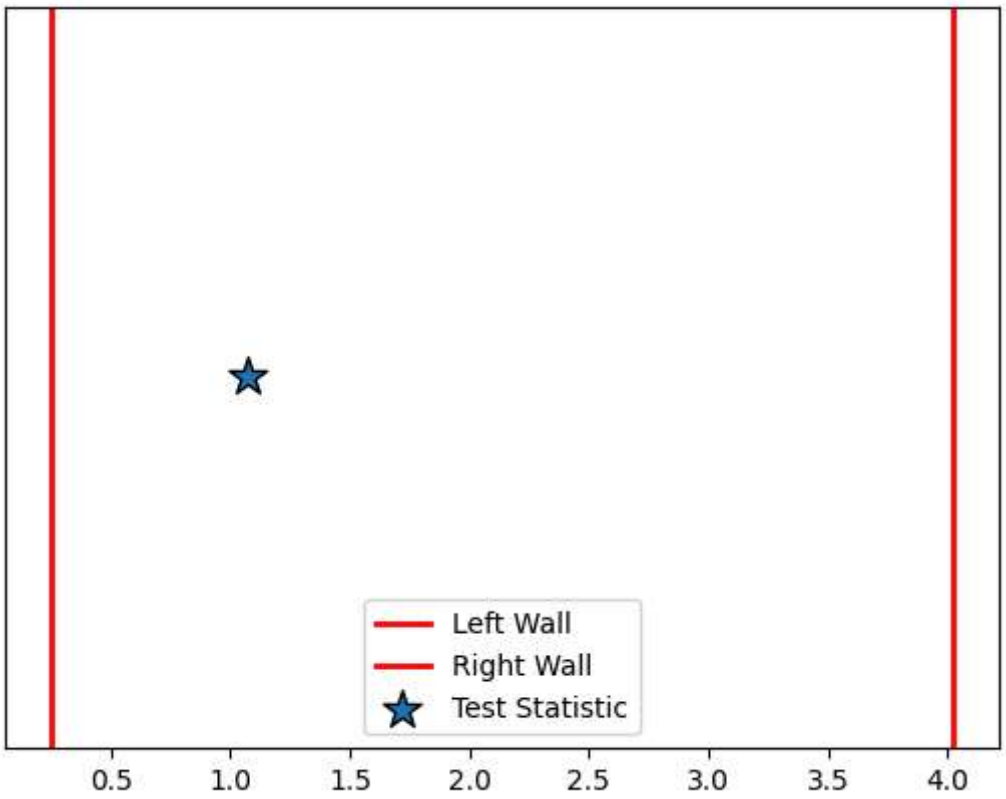
```
test statistic:  19.633
Chi value at 5% level of significance:  9.488
p-value:  0.001
X do not follows Binomial Distribution.
____________________Using inbuilt function____________________
test statistic:  19.633
p-value:  0.001
```

In [ ]:

```
# 20. WAP to test significance of two sample variance.

#                             Ho:  There ratio of two sample variance is not significantly different.
#                             H1:  There ratio of two sample variance is significantly different.
#                     At 5% level of significance.

from scipy.stats import f

def f_test(sample1,sample2):
    n1=len(sample1)
    n2=len(sample2)
    var1=np.var(sample1,ddof=1)
    var2=np.var(sample2,ddof=1)
    if var1>var2:
        ts=var1/var2
        left_wall=f.ppf(0.025,n1-1,n2-1)
        right_wall=f.ppf(0.975,n1-1,n2-1)
    else:
        ts=var2/var1
        left_wall=f.ppf(0.025,n2-1,n1-1)
        right_wall=f.ppf(0.975,n2-1,n1-1)

    print("Test Statistic: ",round(ts,4))
    print("Left Wall and Right Wall : ",[round(left_wall,4),round(right_wall,4)])
```

```
    if left_wall<ts and ts<right_wall:
        print("We do not have sufficient reason to reject Ho at 5% level of significance.")
    else:
        print("We have sufficient reason to reject Ho at 5% level of significance.")

    return


sample1=[27,32.2,30.4,28,26.5,25.5,29.6,27.2]
sample2=[31.4,29.9,33.2,34.4,32,28.7,26.1,30.3]

f_test(sample1,sample2)

plt.axvline(left_wall,color="red",lw=2,label="Left Wall")
plt.axvline(right_wall,color="red",lw=2,label="Right Wall")
plt.scatter(ts,0,marker="*",alpha=1,edgecolor="black",s=200,label="Test Statistic")
plt.gca().get_yaxis().set_visible(False)
plt.legend()
plt.show()
```

```
Test Statistic:  1.3584
Left Wall and Right Wall :  [0.2002, 4.9949]
We do not have sufficient reason to reject Ho at 5% level of significance.
```



In [12]:
```
# 21. WAP to implement linear regression in python.

import statsmodels.api as sm
x=np.array([2,4,7,15,8,10,23,24,18,30])
y=np.array([8,12,14,45,20,23,50,80,65,85])
x=sm.add_constant(x)

model=sm.OLS(y,x)
result=model.fit()
print(result.summary())

x_pred=np.array([3,6,7,2,1,8,21,14,30,24])
x_pred_with_constant=sm.add_constant(x_pred)
y_pred=result.predict(x_pred_with_constant)
```

```python
plt.scatter(x[:, 1], y, color="blue", label="Data Points")
plt.plot(x_pred, y_pred, color="red", label="Regression Line")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.show()
```

```
c:\Users\vasud\AppData\Local\Programs\Python\Python312\Lib\site-packages\scipy\stats\_axis_nan_policy.py:531: UserWarning: kurt
osistest only valid for n>=20 ... continuing anyway, n=10
  res = hypotest_fun_out(*samples, **kwds)
```

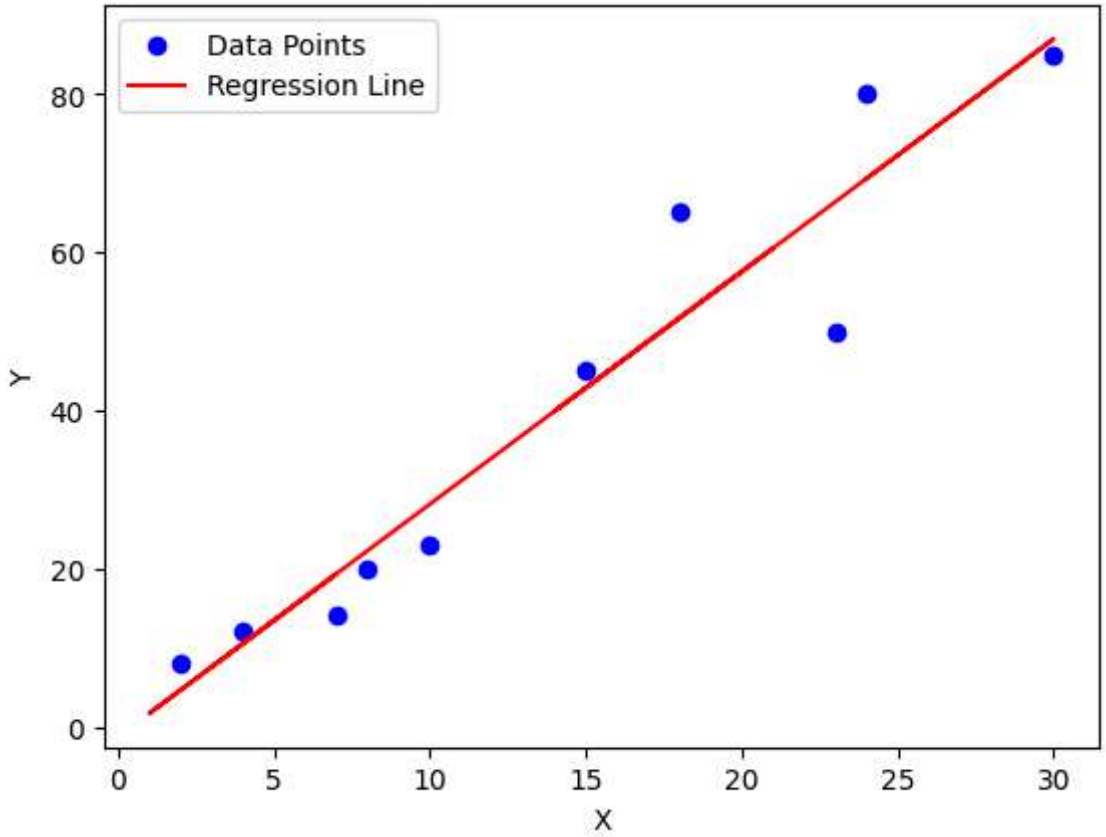
OLS Regression Results

Dep. Variable:	y	R-squared:	0.915
Model:	OLS	Adj. R-squared:	0.904
Method:	Least Squares	F-statistic:	86.05
Date:	Thu, 21 Nov 2024	Prob (F-statistic):	1.48e-05
Time:	11:15:09	Log-Likelihood:	-35.000
No. Observations:	10	AIC:	74.00
Df Residuals:	8	BIC:	74.60
Df Model:	1		
Covariance Type:	nonrobust		

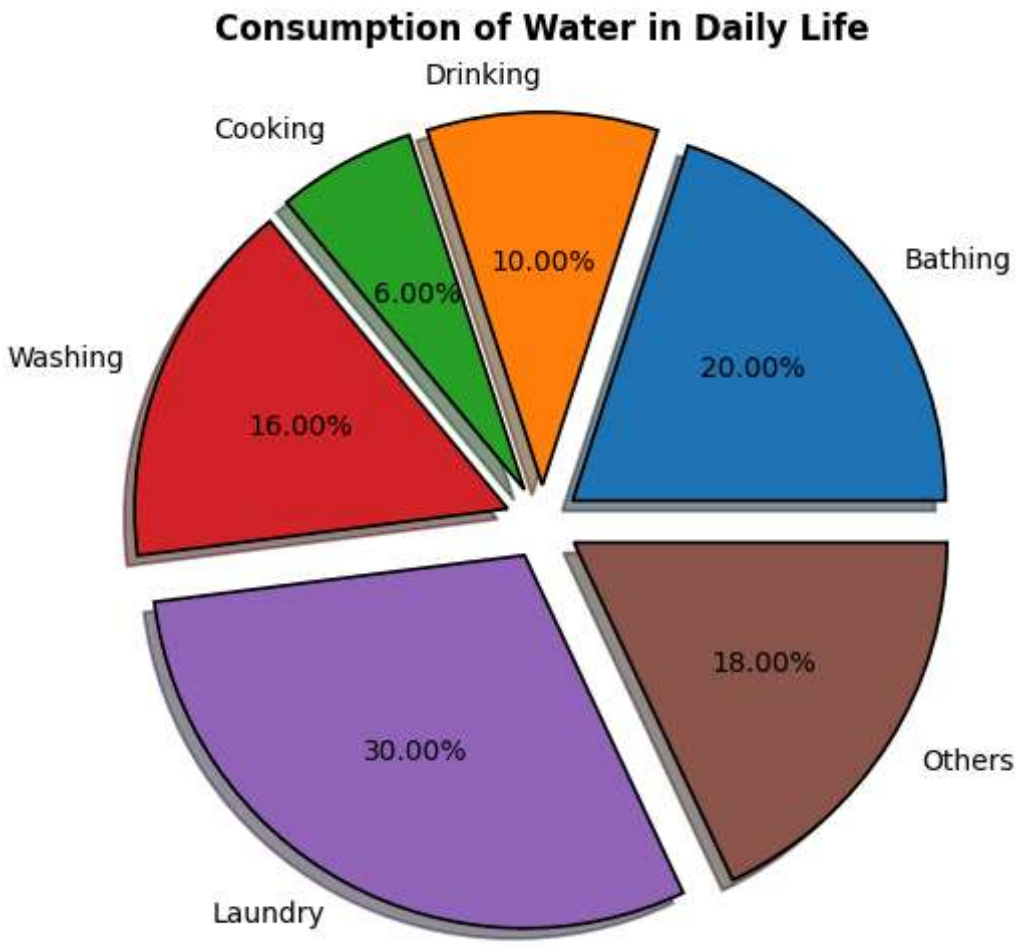
	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.2546	5.291	-0.237	0.819	-13.456	10.947
x1	2.9400	0.317	9.276	0.000	2.209	3.671

Omnibus:	0.652	Durbin-Watson:	1.920
Prob(Omnibus):	0.722	Jarque-Bera (JB):	0.076
Skew:	-0.203	Prob(JB):	0.963
Kurtosis:	2.866	Cond. No.	31.3

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

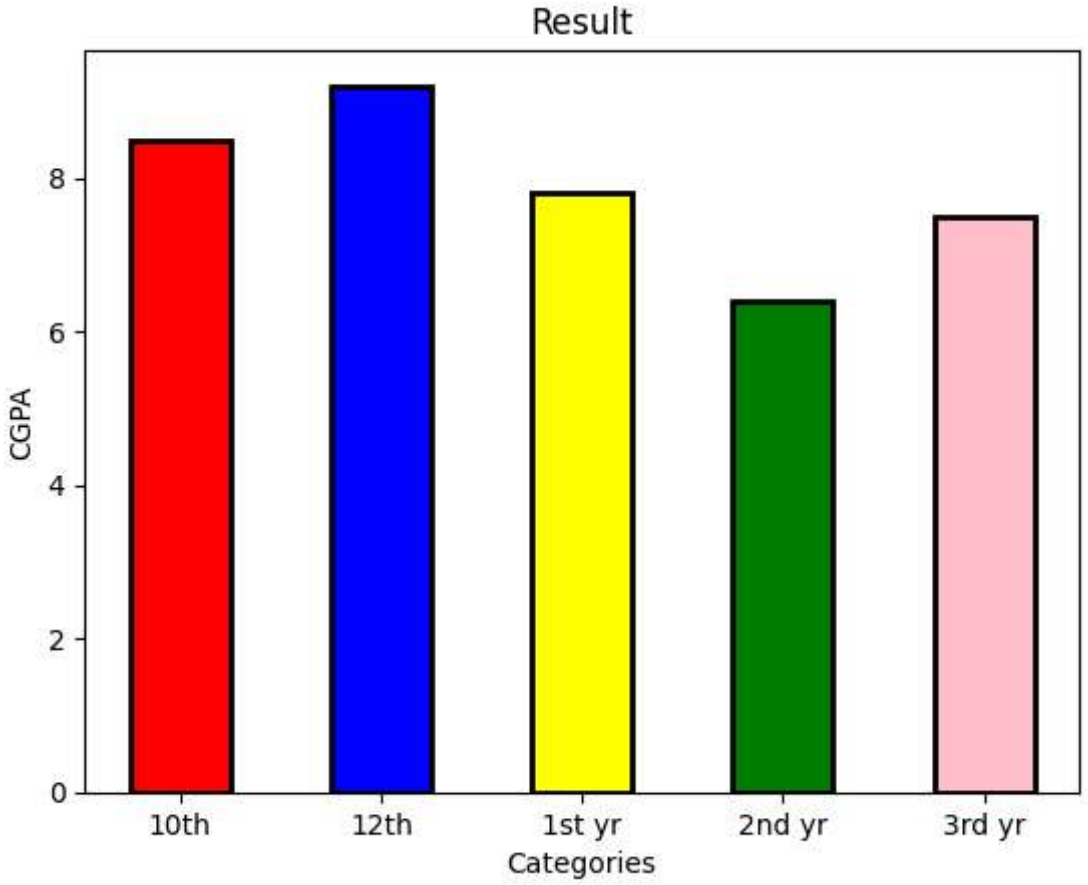


In [ ]:
```
# 22. WAP to plot piechart on consumption of water in daily life.

activities=["Bathing","Drinking","Cooking","Washing","Laundry","Others"]
usage_percentage=[20,10,6,16,30,18]
explode=[0.1,0.1,0.1,0.1,0.1,0.1]

plt.figure(figsize=(6,8))
plt.pie(usage_percentage,labels=activities,explode=explode,autopct="%1.2f%%",shadow=True,wedgeprops={"edgecolor":"black","line
plt.title("Consumption of Water in Daily Life",fontweight='bold')
plt.show()
```



In [ ]:
```
# 23. WAP to plot bar chart to display result for 10th, 12th, 1st year, 2nd year, 3rd year CGPA.

categories=["10th","12th","1st yr","2nd yr","3rd yr"]
cgpa=[8.5,9.2,7.8,6.4,7.5]
color=["red","blue","yellow","green","pink"]

plt.bar(categories,cgpa,width=0.5,color=color,edgecolor="black",lw=2)
plt.xlabel("Categories")
plt.ylabel("CGPA")
```

```
plt.title("Result")
plt.show()
```



In [36]:
```
# 24. WAP to perform various statistical measures using pandas.

np.random.seed(45)
x=np.random.randint(0,100,50)

df=pd.DataFrame(x)

describe=df.describe()
describe.rename(columns={0:"Values"},index={'25%':"Q1","50%":"Median","75%":"Q3"},inplace=True)
describe
```

Out[36]:

	Values
count	50.000000
mean	44.980000
std	26.608645
min	3.000000
Q1	18.750000
Median	44.000000
Q3	64.500000
max	95.000000

In [ ]:

```
# 25. WAP to perform read and write operations with csv files.

def write_tocsv(file_name,data):
    with open(f"{file_name}.csv","w") as data_file:
        for value in data:
            data_file.write(value+'\n')

def read_csv(file_name):
    with open(f"{file_name}.csv","r") as data_file:
        value = data_file.read()
        print(value)

data=["Name,Age",'A,24', 'B,18', 'C,37', 'D,28']

write_tocsv("myfile",data)

read_csv("myfile")
```

```
Name,Age
A,24
B,18
C,37
D,28
```

In [104…

```python
# 26. WAP to compute values of Sin x using taylor series.

import math

def sin_fun(x,terms=10):
    x=math.radians(x)
    sinx=0

    for i in range(terms):
        value=((-1)**i)*(x**((2*i)+1))/math.factorial((2*i)+1)
        sinx+=value

    return sinx

x=90
print(f"sin({x}\N{Degree sign}) equals to {sin_fun(x)}.")
```

```
sin(90°) equals to 1.0.
```

In [ ]:

```python
# 27. WAP to display the following pattern:
#     5                                 00005
#    45                                 00045
#   345      =========>>>>>>>           00345
#  2345                                02345
# 12345                                12345

n=int(input("Enter n: "))

for i in range(0,n):
    for j in range(0,n):
            if i+j<n-1:
                print(" ",end="")
            else:
                print(j+1,end="")
```

```
    print()
```

```
    5
   45
  345
 2345
12345
```

In [ ]:
```
# 28. WAP to find if a number or string is palindrome or not.

def palindrome(x):
    i=0
    j=len(x)-1
    stop=int(len(x)/2)+1
    for a in range(0,stop):
        if i==j and x[i]==x[j]:
            print("Its a Palindrome.")
            return
        elif x[i]==x[j]:
            i+=1
            j-=1
        else :
            print("It's not a Palindrome.")
            return

x1="civic"
palindrome(x1)
x2=[1,2,3,2,1]
palindrome(x2)
```

```
Its a Palindrome.
Its a Palindrome.
```

In [ ]:
```
# 29. WAP to find greatest of number using loop.

def greatest(x):
    i=0
    j=len(x)-1
    for a in range(0,len(x)):
        if x[i]>x[j]:
```

```
            max=x[i]
            j-=1
        else :
            max=x[j]
            i+=1
    print("Greatest Number: ",max)
    return

x=[1,2,9,8,2,4]
greatest(x)
```

```
Greatest Number:  9
```

In [51]:
```
# 30. WAP to print fibonacci series.

def fibonacci(n):
    a=1
    b=2
    if n==0:
        print(n)
    elif n==1:
        return a
    elif n==2:
        return b
    else:
        return fibonacci(n-1)+fibonacci(n-2)

n=int(input("Enter the number of elements: "))

print(f"The Fibonacci Series of first {n} elements is: ")
for i in range(1,n+1):
    print(fibonacci(i),end=" ")
```

```
The Fibonacci Series of first 8 elements is:
1 2 3 5 8 13 21 34
```

In [86]:
```
# 31. WAP to find factorial using recursion.

def factorial(n):
    if n==0 or n==1:
        return 1
```

```
    else:
        return n*factorial(n-1)

n=int(input("Enter the number: "))
print(f"{n}! equals to {factorial(n)}.")
```

```
6! equals to 720.
```

In [ ]:
```
#32. WAP to find if a number is armstrong or not.

n=input('Enter the number: ')
sum=0

for i in n:
    ele=int(i)**len(n)
    sum+=ele

if n==str(sum):
    print(f"{n} is Armstrong.")
else:
    print(f"{n} is not an Armstrong.")
```

```
8208 is Armstrong.
```

In [1]:
```
# 33. Write a menu driven program to find the reverse of a number and sum of digits.

def reverse(n):
    return n[::-1]

def add(n):
    sum=0
    for i in n:
        sum+=int(i)
    return sum

n=input("Enter number: ")
option=input("Press 1:Reverse of a Number, 2:Addition of Digits: ")

print(f"You entered the number {n}.")
if option=="1":
    print(f"Reverse of the number is: {reverse(n)}.")
```

```
elif option=="2":
    print(f"Sum of the digits is: {add(n)}.")
else:
    print("Please select the valid option.")
```

```
You entered the number 67654.
Reverse of the number is: 45676.
```

In [ ]:

In [ ]: