Department of Operational Research
University of Delhi

Python Practical File

Submitted for the course
105 - Python Programming

SUBMITTED BY
Om Gupta
Exam Roll No.: 24236761066
761: Master of Operational Research
North Department
Section - A

SUBMITTED TO
Dr. Gurjeet Kaur
Assistant Professor
Department of Operational Research

Table of Contents

1 Practical Questions

1.1 WAP to compute the roots of a quadratic equation.

[164]:
```
a = float(input("Enter the coefficient of x\N{SUPERSCRIPT TWO}: "))
if a==0:
    print("Not a quadratic equation.\nThe coefficient of x\N{SUPERSCRIPT TWO},␣
 ↪needs to be non-zero.")
else:
    b = float(input("Enter the coefficient of x: "))
    c = float(input("Enter the constant term: "))
    # Formatting the equation for maximum readability.
    print(f"Given equation is: {"" if a==1 else a}x\N{SUPERSCRIPT TWO}{"" if␣
 ↪b==0 else str(" + " if b>0 else " - ")+str(abs(b))+"x"} {"" if c==0 else␣
 ↪str("+ " if c>0 else "- ")+str(abs(c))}")
    from math import sqrt
    # Discriminant
    d = b*b-4*a*c
    if d>0:
        d = sqrt(d)
        root1 = (-b+d)/(2*a)
        root2 = (-b-d)/(2*a)
        print(f"Roots are x = {root1:2.2f}, {root2:2.2f}")
    elif d == 0:
        d = sqrt(d)
        root1 = (-b+d)/(2*a)
        print(f"Root is x = {root1:2.2f}")
    else:
        print(f"Since, discriminant ({d}) < 0\nTherefore, no real roots exist.")
```

```
Given equation is: x² + 1.0x
Roots are x = 0.00, -1.00
```

1.2 WAP to play Stone, Paper, Scissors with Computer.

[22]:
```
u = int(input(f"1. Stone\n2. Paper\n3. Scissors\nEnter your choice (1/2/3): "))
if u>3 or u<1:
    print("Incorrect Choice. Please choose from 1/2/3.")
else:
    from random import randint
    c = randint(1,3)
    l = ['Stone', 'Paper', 'Scissors']
    print(f"You Picked -> {l[u-1]}\nComputer Picked -> {l[c-1]}")
    print("\nResult -> ",end="")
    if u==c:
        print("Draw")
    elif (u==1 and c==3) or (u==2 and c==1) or (u==3 and c==2):
        print("You Won!")
```

```
    else:
        print("Computer Won!")
```

```
You Picked -> Paper
Computer Picked -> Stone

Result -> You Won!
```

1.3 Write a program for BMI Calculator with Categorization of underweight, normal weight and overweight.

BMI Categories
Underweight: $BMI < 18.5$
Normal weight: $18.5 <= BMI < 24.9$
Overweight: $25 <= BMI < 29.9$
Obesity: $BMI >= 30$

[23]:
```
# Input: weight (kg) and height (m)
weight = float(input("Enter your weight in kg: "))
height = float(input("Enter your height in meters: "))

print(f"Weight: {weight}kg")
print(f"Height: {height}m")
bmi = weight / (height ** 2)

print(f"Your BMI is: {bmi:.2f}")

# Categorization
if bmi < 18.5:
    print("You are underweight.")
elif 18.5 <= bmi < 24.9:
    print("You have a normal weight.")
elif 25 <= bmi < 29.9:
    print("You are overweight.")
else:
    print("You are obese.")
```

```
Weight: 70.0kg
Height: 169.5m
Your BMI is: 0.00
You are underweight.
```

1.4 WAP to demonstrate exception handling of Zero Division Error.

[24]:
```
a = int(input("Enter Dividend: "))
b = int(input("Enter a Divisor: "))

try:
```

```
    c = a/b
    print(f"{a}/{b} = {c}")
except ZeroDivisionError as e:
    print("Division by 0 is not possible.")
```

```
Division by 0 is not possible.
```

1.5 WAP to demonstrate OOPs using user defined Cuboid Class.

[25]:
```
class Cuboid:
    def __init__(self, l, b, h):
        self.ln = l
        self.br = b
        self.hi = h
    def peri(self):
        """Compute perimeter of the cuboid."""
        return 4*(self.ln+self.br+self.hi)
    def vol(self):
        """Compute volume of the cuboid."""
        return self.ln*self.br*self.hi
c1 = Cuboid(1,2,3)
perimeter = c1.peri()
volume = c1.vol()
print(f'Perimeter is {perimeter}')
print(f'Volume is {volume}')
```

```
Perimeter is 24
Volume is 6
```

1.6 WAP to demonstrate inheritance in OOPs using user defined Rectangle and Cuboid Class.

[26]:
```
class Rectangle:
    def __init__(self, l, b):
        self.ln = l
        self.br = b
    def peri(self):
        """Compute perimeter of the rectangle."""
        return 2*(self.ln+self.br)
    def area(self):
        """Compute area of the rectangle."""
        return self.ln*self.br

class Cuboid(Rectangle):
    def __init__(self, l, b, h):
        super().__init__(l, b)
        self.hi = h
    def vol(self):
```

```
        """Compute volume of the cuboid."""
        return self.area()*self.hi
    def peri(self):
        """Compute perimeter of the cuboid."""
        return 4*(self.ln+self.br+self.hi)

c2 = Cuboid(3,2,3)
perimeter = c2.peri()
volume = c2.vol()
print(f'Perimeter is {perimeter}')
print(f'Volume is {volume}')
```

```
Perimeter is 32
Volume is 18
```

1.7 Write a Program to implement Inheritance. Create a class Employee inherit two classes Manager and Clerk from Employee.

[27]:
```
class Employee:
    def __init__(self, empid, name, age, salary):
        self.empid = empid
        self.name = name
        self.age = age
        self.salary = salary

    def display_info(self):
        print(f"Employee ID: {self.empid}")
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Salary: {self.salary}")

class Manager(Employee):
    def __init__(self, empid, name, age, salary, department):
        super().__init__(empid, name, age, salary)
        self.department = department

    def display_manager_info(self):
        self.display_info()
        print(f"Department: {self.department}")
        print("Position: Manager")

class Clerk(Employee):
    def __init__(self, empid, name, age, salary, experience):
        super().__init__(empid, name, age, salary)
        self.experience = experience

    def display_clerk_info(self):
```

```python
        self.display_info()
        print(f"Experience: {self.experience} years")
        print("Position: Clerk")

manager = Manager(32347 ,"Rahul", 45, 120000, "HR")
clerk = Clerk(12345, "Ashish", 30, 40000, 5)

print("Manager Details:")
manager.display_manager_info()

print("\nClerk Details:")
clerk.display_clerk_info()
```

```
Manager Details:
Employee ID: 32347
Name: Rahul
Age: 45
Salary: 120000
Department: HR
Position: Manager

Clerk Details:
Employee ID: 12345
Name: Ashish
Age: 30
Salary: 40000
Experience: 5 years
Position: Clerk
```

1.8 WAP to demonstrate the demarcation of class variable and instance variable in OOP using Employee Class.

[28]:
```python
class Employee:
    # Class variable: shared by all instances
    company_name = "Bajaj Tech"
    emp_count = 0

    def __init__(self, name, age, salary):
        # Instance variables: unique to each instance
        self.name = name
        self.age = age
        self.salary = salary
        Employee.emp_count += 1 # Accessing class variable

    def display_info(self):
        """Display information of the Employee."""
        print(f"Employee Name: {self.name}")
```

```python
        print(f"Employee Age: {self.age}")
        print(f"Employee Salary: {self.salary}")
        print(f"Company: {Employee.company_name}")  # Accessing class variable


emp1 = Employee("Rahul", 45, 120000)
emp2 = Employee("Ashish", 30, 40000)

print("Employee 1 Details:")
emp1.display_info()

print("\nEmployee 2 Details:")
emp2.display_info()

# Accessing the class variable directly from the class
print(f"\nNumber of Employees (Accessed via Class): {Employee.emp_count}")

# Modifying the class variable
Employee.company_name = "New Bajaj Tech"

# Display information again after modifying the class variable
print("\nAfter changing the company name for all employees:\n")

print("Employee 1 Details:")
emp1.display_info()

print("\nEmployee 2 Details:")
emp2.display_info()
```

```
Employee 1 Details:
Employee Name: Rahul
Employee Age: 45
Employee Salary: 120000
Company: Bajaj Tech

Employee 2 Details:
Employee Name: Ashish
Employee Age: 30
Employee Salary: 40000
Company: Bajaj Tech

Number of Employees (Accessed via Class): 2

After changing the company name for all employees:

Employee 1 Details:
Employee Name: Rahul
Employee Age: 45
```

```
Employee Salary: 120000
Company: New Bajaj Tech

Employee 2 Details:
Employee Name: Ashish
Employee Age: 30
Employee Salary: 40000
Company: New Bajaj Tech
```

1.9 Write a Program to determine EOQ using various inventory models.

[146]:

```
from abc import ABC, abstractmethod
from math import sqrt
class DetModels(ABC):
    @abstractmethod
    def get_quantity():
        pass
    @abstractmethod
    def get_total_cost():
        pass
    def get_cycle_time(self, Q, lam):
        return Q/lam

class EOQ(DetModels):
    def get_quantity(self, A, lam, I, C):
        return int(sqrt((2*A*lam)/(I*C)))
    def get_total_cost(self, A, lam, I, C, Q):
        holding_cost = (I * C * (Q / 2))
        ordering_cost = A * (lam / Q)
        total_cost = holding_cost + ordering_cost
        return total_cost

class EPQ(DetModels):
    def get_quantity(self, A, lam, I, C, si):
        return int(sqrt((2*A*lam*si)/(I*C*(si-lam))))
    def get_total_cost(self, A, lam, I, C, Q, si):
        holding_cost = (I * C * Q * (si - lam)) / (2 * si)
        setup_cost = A * (lam / Q)
        total_cost = holding_cost + setup_cost
        return total_cost

class EOQ_Short(DetModels):
    def get_quantity(self, A, lam, I, C, pi):
        return int(sqrt((2*A*lam*(pi + I*C))/(I*C*pi)))
    def get_total_cost(self, A, lam, I, C, Q, pi):
        Q_r = Q * (I * C) / (pi + I * C)
        holding_cost = I * C * (Q / 2)
```

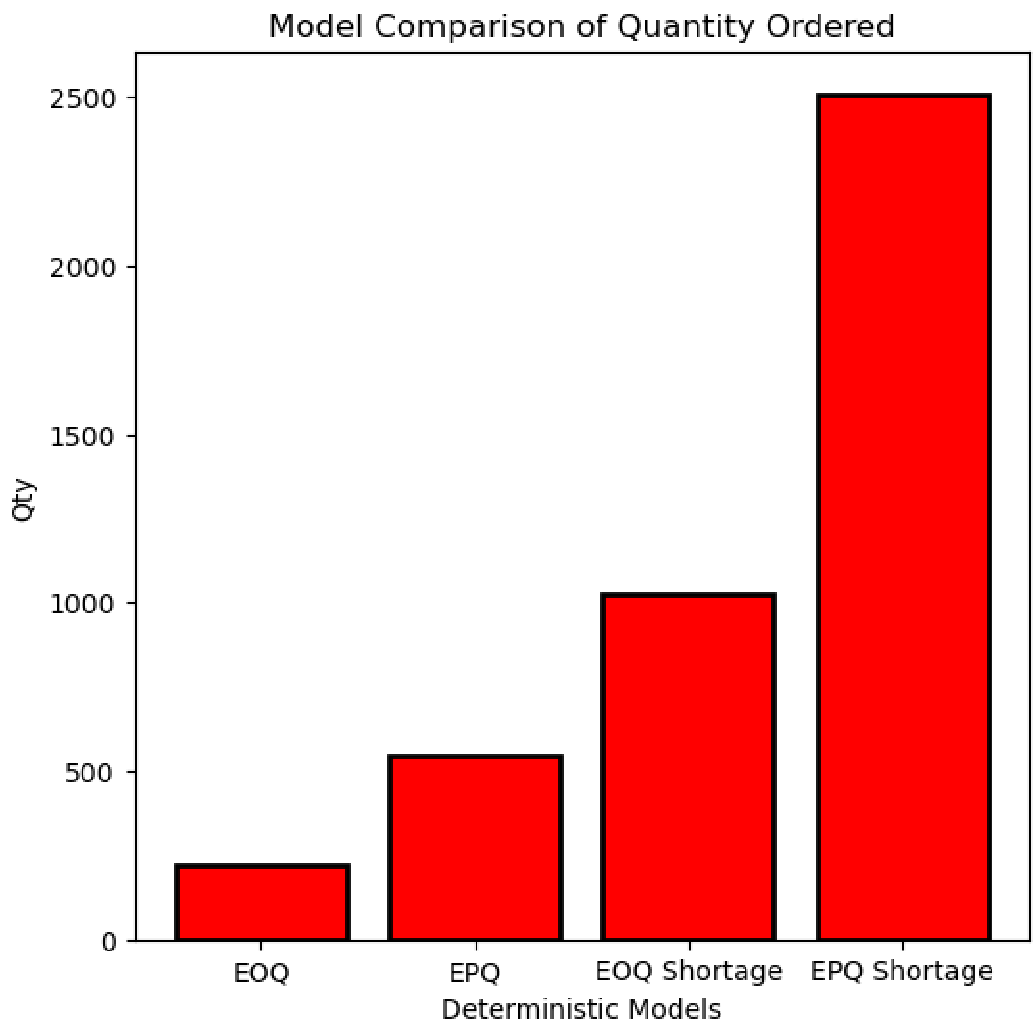
```
        ordering_cost = A * (lam / Q)
        shortage_cost = pi * (Q - Q_r) / 2
        total_cost = holding_cost + ordering_cost + shortage_cost
        return total_cost

class EPQ_Short(DetModels):
    def get_quantity(self, A, lam, I, C, si, pi):
        return int(sqrt((2*A*lam*(pi + I*C)*si)/(I*C*pi*(si-lam))))
    def get_total_cost(self, A, lam, I, C, Q, si, pi):
        Q_r = Q * (I * C) / (pi + I * C)
        holding_cost = (I * C * Q * (si - lam)) / (2 * si)
        setup_cost = A * (lam / Q)
        shortage_cost = pi * (Q - Q_r) / 2
        total_cost = holding_cost + setup_cost + shortage_cost
        return total_cost
```

[147]:
```
# Sample Run
A = float(input("Enter ordering cost per order (A): ")) # 100
lam = float(input("Enter demand rate (lambda): ")) # 10000
I = float(input("Enter inventory carrying cost rate (I): ")) # 0.2
C = float(input("Enter unit cost of item (C): ")) # 200

print("\nRunning for EOQ")
my_eoq = EOQ()
Q = my_eoq.get_quantity(A, lam, I, C)
tc = my_eoq.get_total_cost(A, lam, I, C, Q)
T = my_eoq.get_cycle_time(Q, lam)
print(f"Quantity = {Q} units")
print(f"Total Cost =  {tc:2.2f}")
print(f"Cycle Time = {T*365:2.2f} days")

si = float(input("Enter production rate (s): ")) # 12000
print("\nRunning for EPQ")
my_epq = EPQ()
Q2 = my_epq.get_quantity(A, lam, I, C, si)
tc = my_epq.get_total_cost(A, lam, I, C, Q, si)
T = my_epq.get_cycle_time(Q, lam)
print(f"Quantity = {Q2} units")
print(f"Total Cost =  {tc:2.2f}")
print(f"Cycle Time = {T*365:2.2f} days")

pi = float(input("Enter shortage cost per unit (p): ")) # 2
print("\nRunning for EOQ with Shortage")
my_eoq_short = EOQ_Short()
Q3 = my_eoq_short.get_quantity(A, lam, I, C, pi)
tc = my_eoq_short.get_total_cost(A, lam, I, C, Q, pi)
T = my_eoq_short.get_cycle_time(Q, lam)
```

```python
print(f"Quantity = {Q3} units")
print(f"Total Cost =  {tc:2.2f}")
print(f"Cycle Time = {T*365:2.2f} days")

print("\nRunning for EPQ with Shortage")
my_epq_short = EPQ_Short()
Q4 = my_epq_short.get_quantity(A, lam, I, C, si, pi)
tc = my_epq_short.get_total_cost(A, lam, I, C, Q, si, pi)
T = my_epq_short.get_cycle_time(Q, lam)
print(f"Quantity = {Q4} units")
print(f"Total Cost =  {tc:2.2f}")
print(f"Cycle Time = {T*365:2.2f} days")
```

```
Running for EOQ
Quantity = 223 units
Total Cost =  8944.30
Cycle Time = 8.14 days

Running for EPQ
Quantity = 547 units
Total Cost =  5227.64
Cycle Time = 8.14 days

Running for EOQ with Shortage
Quantity = 1024 units
Total Cost =  8954.92
Cycle Time = 8.14 days

Running for EPQ with Shortage
Quantity = 2509 units
Total Cost =  5238.26
Cycle Time = 8.14 days
```

[ ]:
```python
# Visualising Q for each model
import matplotlib.pyplot as plt

l = [Q, Q2, Q3, Q4]
names = ["EOQ", "EPQ", "EOQ Shortage", "EPQ Shortage"]

plt.figure(figsize=(6,6))
plt.bar(names, l, color = 'r',edgecolor='k',linewidth=2)
plt.ylabel("Qty")
plt.xlabel("Deterministic Models")
plt.title("Model Comparison of Quantity Ordered")
plt.show()
```



1.10 Write a Program to determine different characteristics using various Queuing models.

[3]:
```
from math import factorial

class MM1:
    def __init__(self, lam, mu):
        self.lam = lam  # Arrival rate
        self.mu = mu    # Service rate
        self.rho = lam / mu  # Traffic intensity

    def get_pn(self, n):
        """ Probability of having n customers in the system """
        return (1 - self.rho) * (self.rho ** n)
```

```python
    def L(self):
        """ Average number of customers in the system """
        return self.rho / (1 - self.rho)

    def Lq(self):
        """ Average number of customers in the queue """
        return (self.rho ** 2) / (1 - self.rho)

    def W(self):
        """ Average time a customer spends in the system """
        return 1 / (self.mu - self.lam)

    def Wq(self):
        """ Average time a customer spends waiting in the queue """
        return self.rho / (self.mu - self.lam)


class MM1k:
    def __init__(self, lam, mu, k):
        self.lam = lam  # Arrival rate
        self.mu = mu    # Service rate
        self.k = k      # Capacity
        self.rho = lam / mu

    def get_p0(self):
        """ Probability of zero customers in the system """
        if self.rho == 1:
            return 1 / (self.k + 1)
        else:
            return (1 - self.rho) / (1 - self.rho ** (self.k + 1))

    def get_pn(self, n):
        """ Probability of having n customers in the system """
        p0 = self.get_p0()
        return p0 * (self.rho ** n)

    def L(self):
        """ Average number of customers in the system """
        p0 = self.get_p0()
        L = 0
        for n in range(self.k + 1):
            L += n * self.get_pn(n)
        return L

    def Lq(self):
        """ Average number of customers in the queue """
        return self.L() - self.rho
```

```python
    def W(self):
        """ Average time a customer spends in the system """
        return self.L() / (self.lam * (1 - self.get_pn(self.k)))

    def Wq(self):
        """ Average time a customer spends waiting in the queue """
        return self.Lq() / (self.lam * (1 - self.get_pn(self.k)))


class MMc:
    def __init__(self, lam, mu, c):
        self.lam = lam  # Arrival rate
        self.mu = mu    # Service rate
        self.c = c      # Number of servers
        self.rho = lam / (c * mu)

    def get_p0(self):
        """ Probability of zero customers in the system """
        summation = sum((self.lam / self.mu) ** n / factorial(n) for n in␣
↪range(self.c))
        last_term = (self.lam / self.mu) ** self.c / (factorial(self.c) * (1 -␣
↪self.rho))
        return 1 / (summation + last_term)

    def get_pn(self, n):
        """ Probability of having n customers in the system """
        p0 = self.get_p0()
        if n < self.c:
            return p0 * (self.lam / self.mu) ** n / factorial(n)
        else:
            return p0 * (self.lam / self.mu) ** n / (factorial(self.c) * self.c␣
↪** (n - self.c))

    def Lq(self):
        """ Average number of customers in the queue """
        p0 = self.get_p0()
        return p0 * (self.rho ** self.c) * self.lam / (factorial(self.c) * (1 -␣
↪self.rho) ** 2)

    def L(self):
        """ Average number of customers in the system """
        return self.Lq() + self.lam / self.mu

    def W(self):
        """ Average time a customer spends in the system """
        return self.L() / self.lam
```

```python
    def Wq(self):
        """ Average time a customer spends waiting in the queue """
        return self.Lq() / self.lam


class MMck:
    def __init__(self, lam, mu, c, k):
        self.lam = lam  # Arrival rate
        self.mu = mu    # Service rate
        self.c = c      # Number of servers
        self.k = k      # Capacity
        self.rho = lam / (c * mu)

    def get_p0(self):
        """ Probability of zero customers in the system """
        sum1 = sum((self.lam / self.mu) ** n / factorial(n) for n in range(self.c))
        sum2 = ((self.lam / self.mu) ** self.c) / factorial(self.c) * sum(
            (self.lam / (self.c * self.mu)) ** n for n in range(self.k - self.c + 1)
        )
        return 1 / (sum1 + sum2)

    def get_pn(self, n):
        """ Probability of having n customers in the system """
        p0 = self.get_p0()
        if n < self.c:
            return p0 * (self.lam / self.mu) ** n / factorial(n)
        else:
            return p0 * (self.lam / self.mu) ** n / (factorial(self.c) * self.c ** (n - self.c))

    def Lq(self):
        """ Average number of customers in the queue """
        p0 = self.get_p0()
        return p0 * (self.rho ** self.c) * self.lam / (factorial(self.c) * (1 - self.rho) ** 2)

    def L(self):
        """ Average number of customers in the system """
        return self.Lq() + self.lam / self.mu

    def W(self):
        """ Average time a customer spends in the system """
        return self.L() / self.lam
```

```python
    def Wq(self):
        """ Average time a customer spends waiting in the queue """
        return self.Lq() / self.lam
```

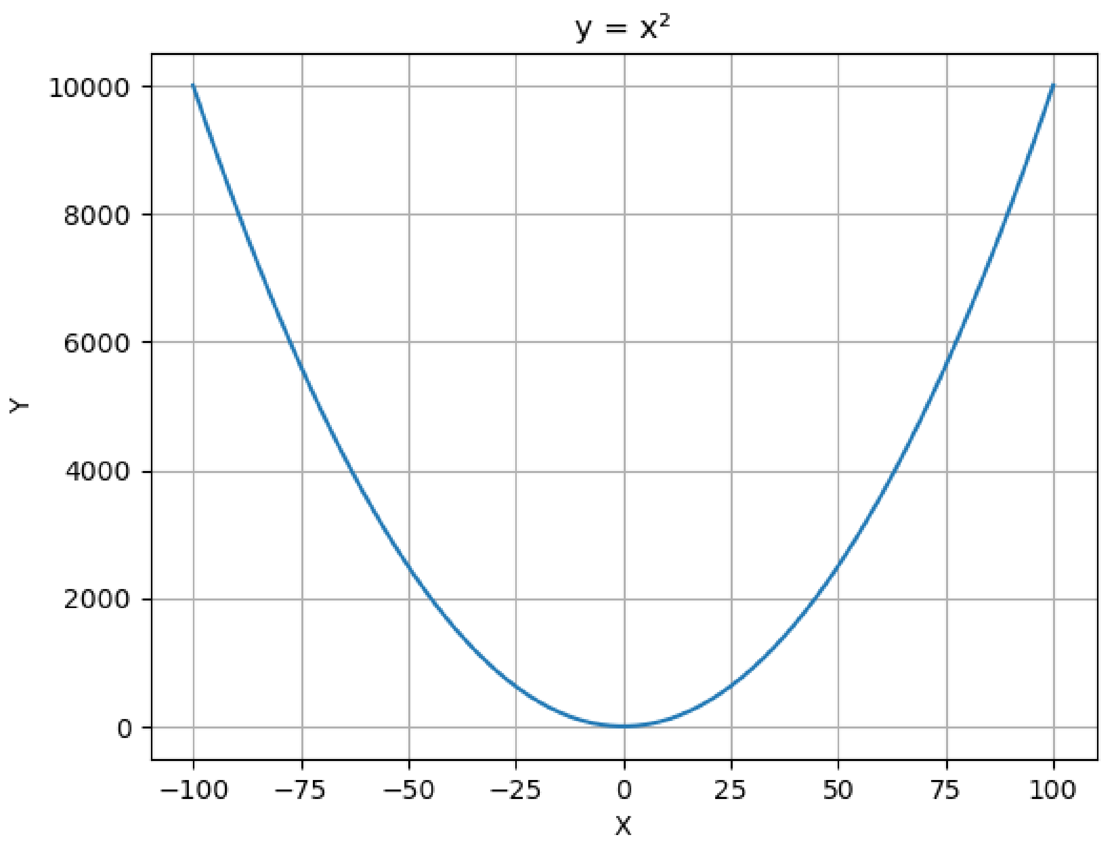
[12]:
```python
# Sample Run
mm1 = MM1(lam=2, mu=3)
print("Model: M/M/1")
print("L:", round(mm1.L(),2))
print("Lq:", round(mm1.Lq(),2))
print("W:", round(mm1.W(),2))
print("Wq:", round(mm1.Wq(),2))

mm1k = MM1k(lam=2, mu=3, k=5)
print("\nModel: M/M/1/k")
print("L:", round(mm1k.L(),2))
print("Lq:", round(mm1k.Lq(),2))
print("W:", round(mm1k.W(),2))
print("Wq:", round(mm1k.Wq(),2))

mmc = MMc(lam=5, mu=6, c=3)
print("\nModel: M/M/c")
print("L:", round(mmc.L(),2))
print("Lq:", round(mmc.Lq(),2))
print("W:", round(mmc.W(),2))
print("Wq:", round(mmc.Wq(),2))

mmck = MMck(lam=4, mu=5, c=2, k=10)
print("\nModel: M/M/c/k")
print("L:", round(mmck.L(),2))
print("Lq:", round(mmck.Lq(),2))
print("W:", round(mmck.W(),2))
print("Wq:", round(mmck.Wq(),2))
```

```
Model: M/M/1
L: 2.0
Lq: 1.33
W: 1.0
Wq: 0.67

Model: M/M/1/k
L: 1.42
Lq: 0.76
W: 0.75
Wq: 0.4

Model: M/M/c
L: 0.85
Lq: 0.01
```

```
W: 0.17
Wq: 0.0

Model: M/M/c/k
L: 1.18
Lq: 0.38
W: 0.3
Wq: 0.1
```

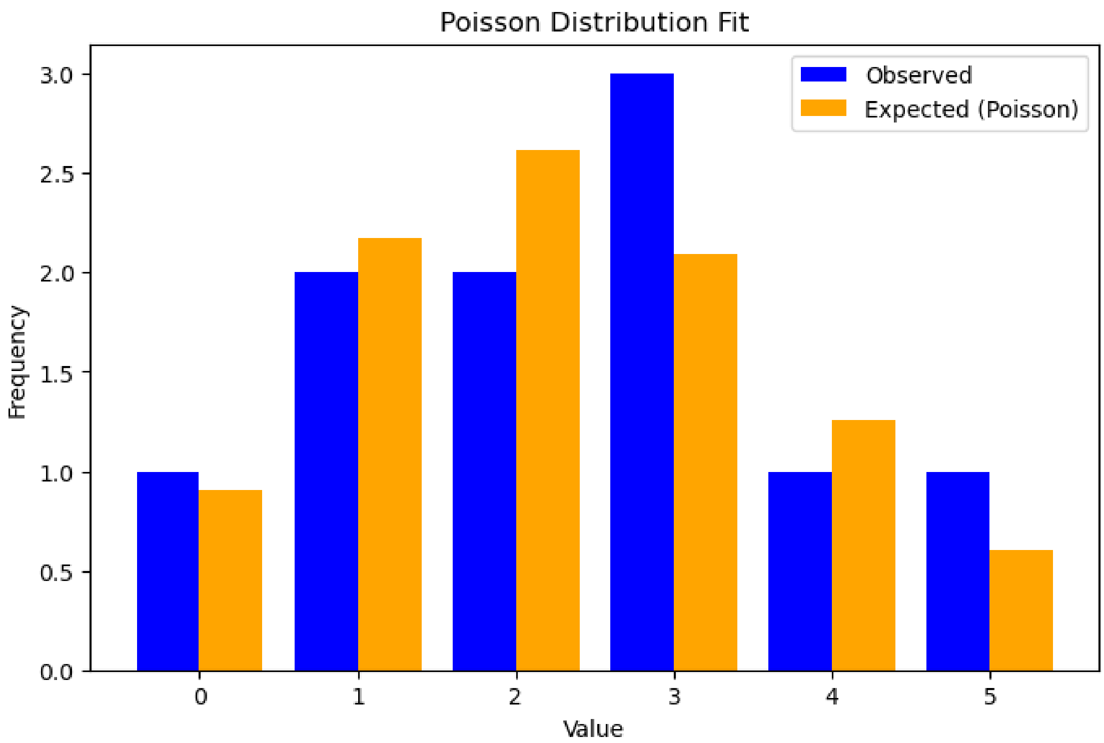
1.11 WAP to plot a graph for function $y = x^2$.

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-100,101)
y = x**2

plt.plot(x,y)
plt.xlabel("X")
plt.ylabel("Y")
plt.title("y = x\N{SUPERSCRIPT TWO}")
plt.grid()
plt.show()
```



1.12 WAP to fit poisson distribution on a given data.

[113]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson

data = [2, 3, 1, 0, 5, 3, 2, 4, 1, 3]

data_mean = np.mean(data)

x = np.arange(0, max(data) + 1)
poisson_probs = poisson.pmf(x, data_mean)

observed_freq = [data.count(i) for i in x]

plt.figure(figsize=(8, 5))
plt.bar(x - 0.2, observed_freq, width=0.4, label="Observed", color="blue")
plt.bar(x + 0.2, poisson_probs * len(data), width=0.4, label="Expected␣
↪(Poisson)", color="orange")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.title("Poisson Distribution Fit")
plt.legend()
plt.show()
```



1.13 Write a python function that calculates the pearson correlation cofficient between two lists of numbers.

[114]:
```
def pearson_correlation(x, y):
    if len(x) != len(y):
        raise ValueError("Lists must have the same length")

    mean_x, mean_y = np.mean(x), np.mean(y)
    numerator = sum((xi - mean_x) * (yi - mean_y) for xi, yi in zip(x, y))
    denominator = np.sqrt(sum((xi - mean_x)**2 for xi in x) * sum((yi -␣
 ↪mean_y)**2 for yi in y))

    return numerator / denominator

# Example
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
print("Pearson Correlation:", pearson_correlation(x, y))
```

```
Pearson Correlation: 1.0
```

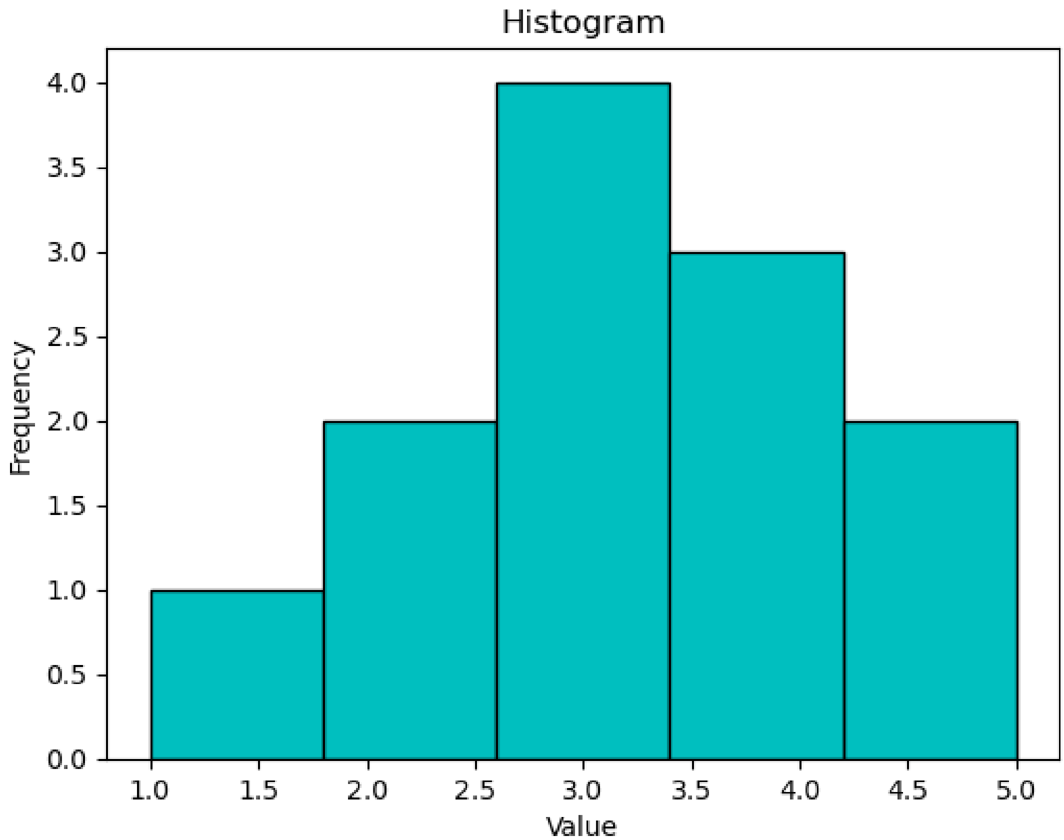
1.14 Write a python function that calculates `Spearman correlation cofficient`.

[115]:
```
from scipy.stats import rankdata

def spearman_correlation(x, y):
    if len(x) != len(y):
        raise ValueError("Lists must have the same length")

    rank_x = rankdata(x)
    rank_y = rankdata(y)

    return pearson_correlation(rank_x, rank_y)

# Example
x = [10, 20, 30, 40, 50]
y = [1, 2, 3, 4, 5]
print("Spearman Correlation:", spearman_correlation(x, y))
```

```
Spearman Correlation: 1.0
```

1.15 Using matplotlib plot histogram of list of numbers.

[15]:
```
data = [1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 5, 5]
plt.hist(data, bins=5, color='c', edgecolor='black')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram')
plt.show()
```



1.16 Write a python function that calculates the Z-score for the lists of numbers.

[117]:
```
def calculate_z_score(data):
    mean = np.mean(data)
    std_dev = np.std(data)
    return [(x - mean) / std_dev for x in data]

# Example
data = [10, 20, 30, 40, 50]
print("Z-Scores:", calculate_z_score(data))
```

```
Z-Scores: [-1.414213562373095, -0.7071067811865475, 0.0, 0.7071067811865475,
1.414213562373095]
```

1.17 WAP to test the significance of two sample means.

[118]:
```
from scipy.stats import ttest_ind

def test_sample_means(sample1, sample2):
    t_stat, p_value = ttest_ind(sample1, sample2, equal_var=False)
```

```
    return t_stat, p_value

# Example
sample1 = [1, 2, 3, 4, 5]
sample2 = [5, 6, 7, 8, 9]
t_stat, p_value = test_sample_means(sample1, sample2)
print(f"T-Statistic: {t_stat}, P-Value: {p_value}")
```

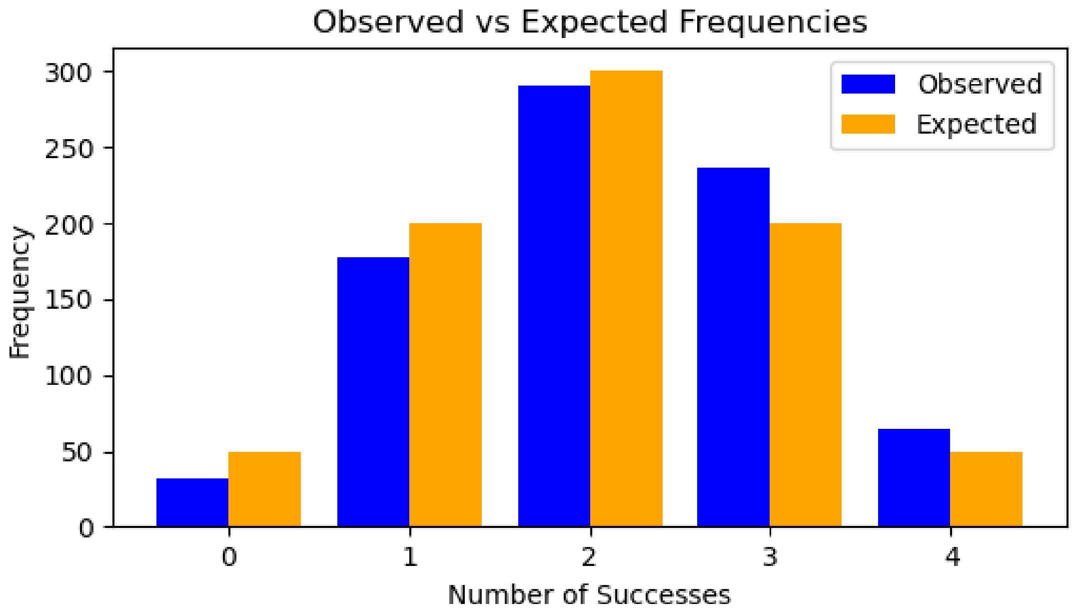
```
T-Statistic: -4.0, P-Value: 0.003949772803445322
```

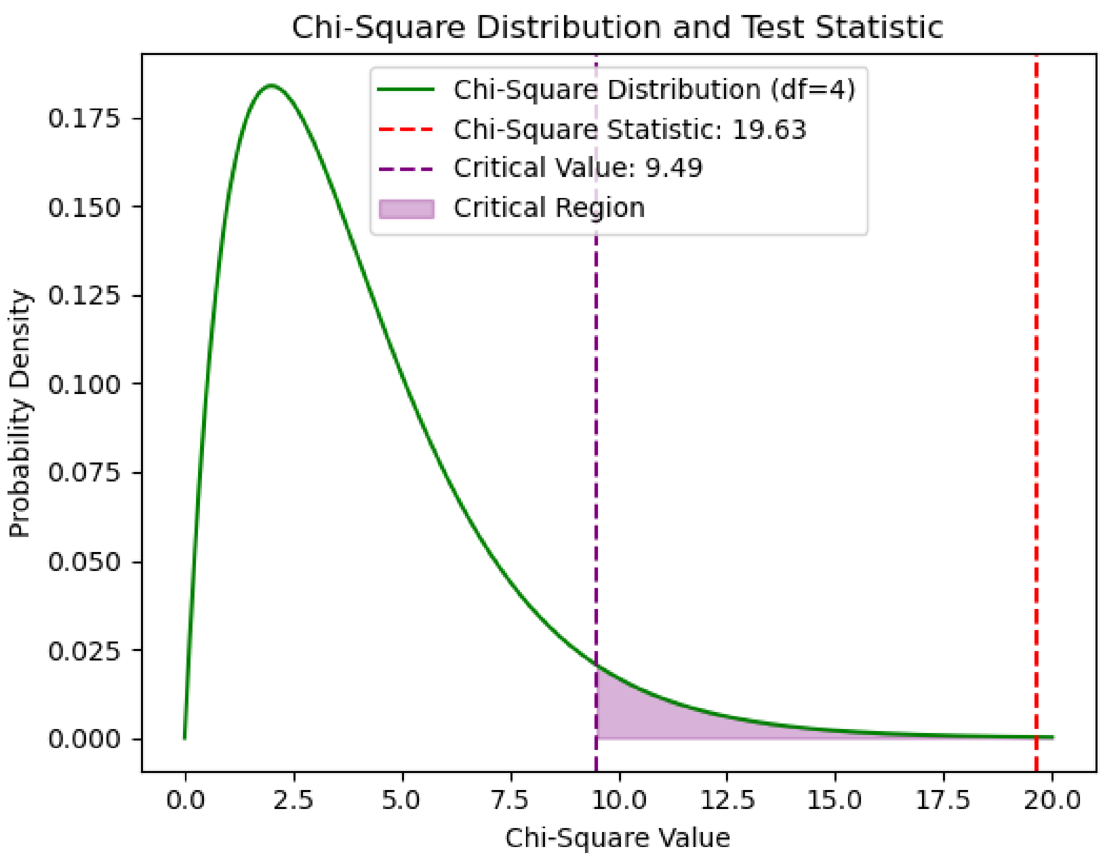
1.18 WAP to test the goodness of fit of a given dataset on binomial distribution.

[143]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom, chisquare, chi2

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

N = o.sum()  # Total number of observations = 800
n = 4         # Number of trials

Pi = binom.pmf(x, n, 0.5) # Expected Frequencies
E = Pi * N

chi_stat, p_value = chisquare(o, E)
chi_tab = chi2.ppf(0.95, df=n)

print(f"Chi-Square Statistic: {chi_stat}")
print(f"P-Value: {p_value}")
print(f"Critical Value: {chi_tab}")

if chi_stat > chi_tab:
    print("Reject the null hypothesis: Observed frequencies do not fit the␣
 ↪expected distribution.")
else:
    print("Fail to reject the null hypothesis: Observed frequencies fit the␣
 ↪expected distribution.")

# plotting
fig, axs = plt.subplots(2, 1, figsize=(6, 8), gridspec_kw={'height_ratios': [2,␣
 ↪3]})

# Plot 1: Observed vs Expected Frequencies (Bar Plot)
axs[0].bar(x - 0.2, o, width=0.4, label='Observed', color='blue',␣
 ↪align='center')
```

```python
axs[0].bar(x + 0.2, E, width=0.4, label='Expected', color='orange', align='center')
axs[0].set_xlabel('Number of Successes')
axs[0].set_ylabel('Frequency')
axs[0].set_title('Observed vs Expected Frequencies')
axs[0].legend()

# Plot 2: Chi-Square Distribution and Test Statistic
x_vals = np.linspace(0, 20, 500)
y_vals = chi2.pdf(x_vals, df=n)

axs[1].plot(x_vals, y_vals, label="Chi-Square Distribution (df=4)", color='green')
axs[1].axvline(chi_stat, color='red', linestyle='--', label=f"Chi-Square Statistic: {chi_stat:.2f}")
axs[1].axvline(chi_tab, color='purple', linestyle='--', label=f"Critical Value: {chi_tab:.2f}")
axs[1].fill_between(x_vals, 0, y_vals, where=(x_vals >= chi_tab), color='purple', alpha=0.3, label='Critical Region')
axs[1].set_xlabel('Chi-Square Value')
axs[1].set_ylabel('Probability Density')
axs[1].set_title('Chi-Square Distribution and Test Statistic')
axs[1].legend()

plt.tight_layout()
plt.show()
```

```
Chi-Square Statistic: 19.633333333333332
P-Value: 0.0005898876209430505
Critical Value: 9.487729036781154
Reject the null hypothesis: Observed frequencies do not fit the expected
distribution.
```


Observed vs Expected Frequencies
Observed
Expected
Frequency
Number of Successes
300
250
200
150
100
50
0
0
1
2
3
4


Chi-Square Distribution and Test Statistic
Chi-Square Distribution (df=4)
Chi-Square Statistic: 19.63
Critical Value: 9.49
Critical Region
Probability Density
Chi-Square Value
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

1.19 WAP to test significance of two sample variance.

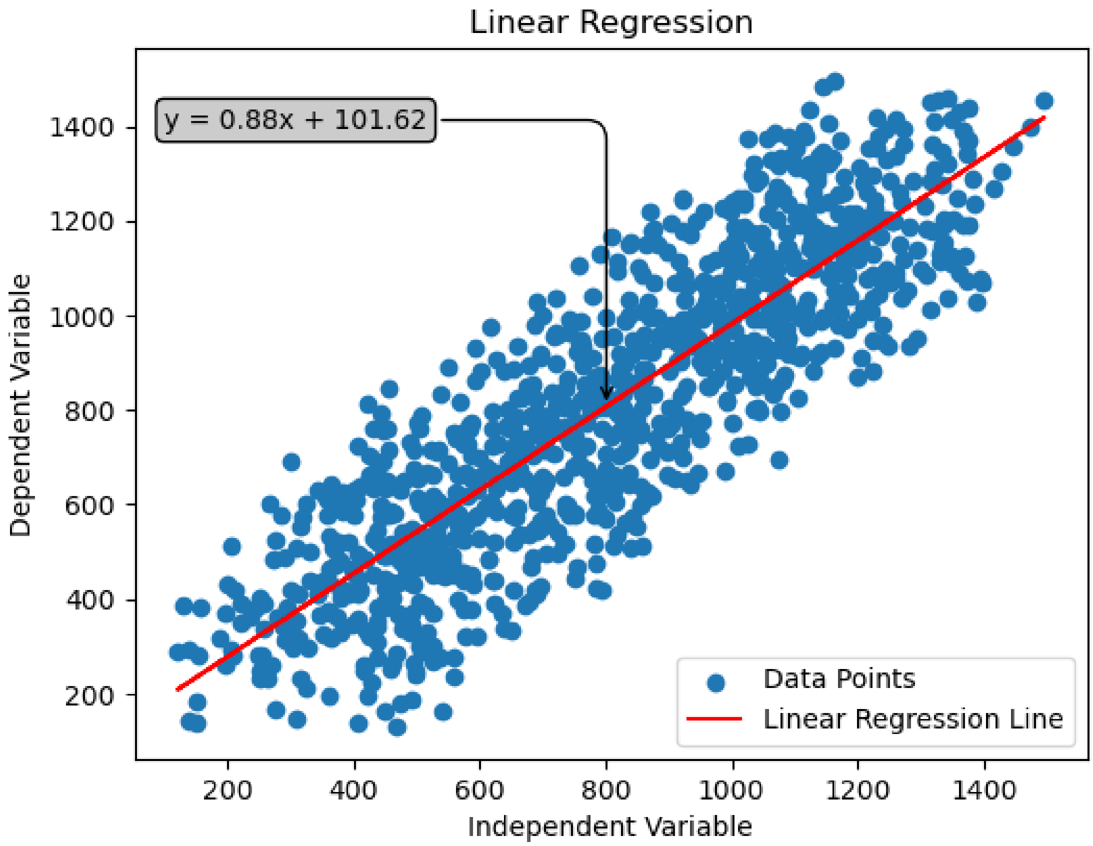
[18]:
```
from scipy.stats import f_oneway

def test_sample_variances(sample1, sample2):
    f_stat, p_value = f_oneway(sample1, sample2)
    return f_stat, p_value

# Example
sample1 = [10, 12, 14, 16, 18]
sample2 = [20, 22, 24, 26, 28]
f_stat, p_value = test_sample_variances(sample1, sample2)
print(f"F-Statistic: {f_stat}, P-Value: {p_value}")

print(f"{'' if p_value < 0.05 else "No "}Significant Difference in variances.")
```

```
F-Statistic: 25.0, P-Value: 0.0010528257933665396
Significant Difference in variances.
```

1.20 WAP to implement linear regression in python.

[37]:
```
import numpy as np
# Generating Random Data Points
n = 1000
x = np.linspace(1,1000, n) + np.random.randint(100,500, n)
y = np.linspace(10,1000, n) + np.random.randint(100,500, n)

m = (n*(x*y).sum() - x.sum()*y.sum())/(n*(x**2).sum()-(x.sum())**2)
b = (y.sum() - m*x.sum())/n
y_ = m*x + b # Equation of Linear Regression Line
```

[70]:
```
import matplotlib.pyplot as plt
plt.scatter(x,y, label = "Data Points")
plt.plot(x,y_, color = 'r', label = "Linear Regression Line")
plt.xlabel("Independent Variable")
plt.ylabel("Dependent Variable")
plt.legend(loc='lower right')
plt.title("Linear Regression")
# Adding Annotation of Equation of linear regression line
plt.annotate(f"y = {m:.2f}x + {b:.2f}",
             xy=(x.mean(), m*x.mean()+b),
             xytext=(100, y.max()-100),
             arrowprops=dict(
                 arrowstyle = "->",
                 connectionstyle = "angle, angleA = 0, angleB = 90, rad = 10"),
            bbox = dict(boxstyle ="round", fc ="0.8"),
            fontsize=10)
plt.show()
```



1.20.1 Explanaiton of Linear Regression Code

In linear regression, the goal is to fit a line that best represents the relationship between a dependent variable y and an independent variable x. The line is represented by the equation:

$$y = \beta_0 + \beta_1 x$$

Where:
β_0 is the y-intercept &
β_1 is the slope of the line.

To estimate β_0 and β_1, we use the method of **Least Squares**, which minimizes the sum of the squared differences between the observed values y_i and the predicted values $\hat{y}_i$ from the line. The formulae for estimating β_0 and β_1 are derived as follows:

1.20.2 Slope β_1:

$$\beta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

The simplified slope formula is: **(Used in Code)**

$$\beta_1 = \frac{n\sum(x_i y_i) - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}$$

Where:
$\bar{x}$ is the mean of the x-values,
$\bar{y}$ is the mean of the y-values &
n is the number of data points.

1.20.3 Intercept β_0:

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

The simplified intercept formula is: **(Used in Code)**

$$\beta_0 = \frac{\sum y_i - \beta_1 \cdot \sum x_i}{n}$$

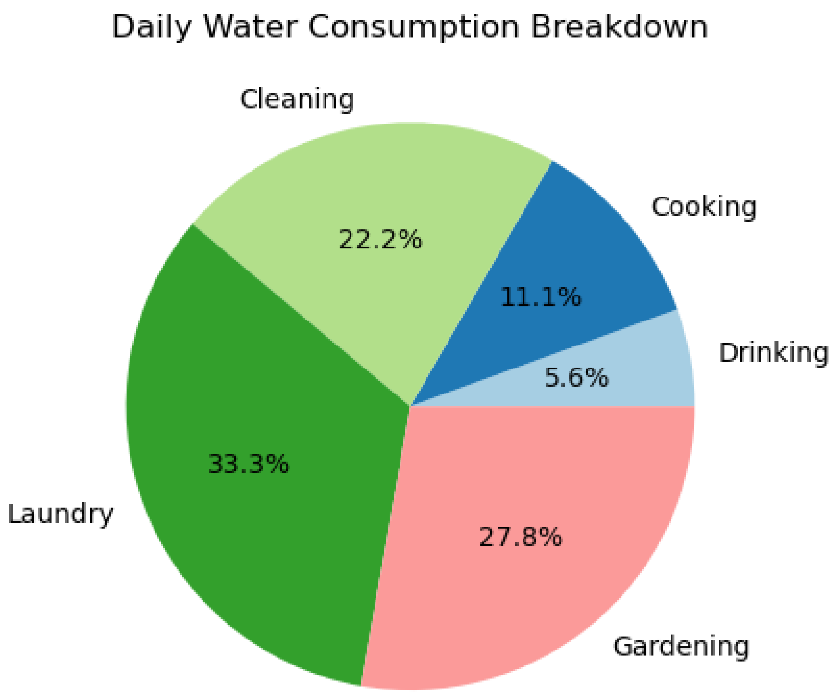
1.20.4 Sum of Squared Errors (SSE):

The total error, which is minimized in this method, is the sum of squared residuals (differences between actual and predicted values):

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}\left(y_i - (\beta_0 + \beta_1 x_i)\right)^2$$

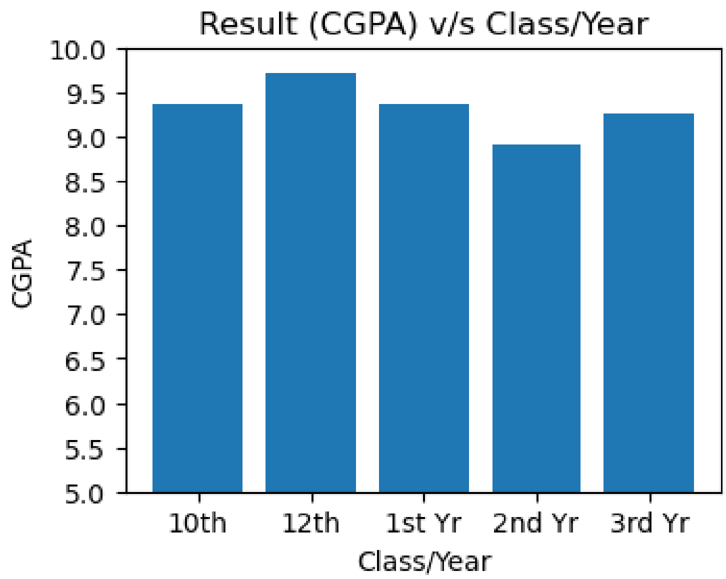
1.21 WAP to plot piechart on consumption of water in daily life.

[76]:
```
import matplotlib.pyplot as plt

activities = ['Drinking', 'Cooking', 'Cleaning', 'Laundry', 'Gardening']
water_consumption = [5, 10, 20, 30, 25]

plt.pie(water_consumption, labels=activities, autopct='%1.1f%%', colors=plt.cm.
 ↪Paired.colors)
plt.title('Daily Water Consumption Breakdown')

plt.show()
```



1.22 WAP to plot bar chart to display result for 10th, 12th, 1st year, 2nd year, 3rd year CGPA.

[26]:
```
results = [9.36, 9.72,9.36, 8.91,9.25]
year = ["10th", "12th", "1st Yr", "2nd Yr", "3rd Yr"]

plt.figure(figsize=(4,3))
plt.bar(year, results)
plt.ylim(5,10)
plt.yticks(np.arange(5,10.1,0.5))
plt.xlabel("Class/Year")
plt.ylabel("CGPA")
plt.title("Result (CGPA) v/s Class/Year")
plt.show()
```



1.23 WAP to perform various statistical measures using pandas.

[ ]:
```
import pandas as pd

# Sample Data
data = {
    'A': [10, 20, 30, 40, 50],
    'B': [5, 15, 25, 35, 45],
    'C': [2, 4, 6, 8, 10]
}
df = pd.DataFrame(data)
df
```

Dataset:

[ ]:

	A	B	C
0	10	5	2
1	20	15	4
2	30	25	6
3	40	35	8
4	50	45	10

[125]:
```
print("Various Statistical Measures")
print("\nMean of columns:\n", df.mean())
print("\nMedian of columns:\n", df.median())
print("\nStandard Deviation of columns:\n", df.std())
print("\nVariance of columns:\n", df.var())
print("\nMinimum of columns:\n", df.min())
```

```python
print("\nMaximum of columns:\n", df.max())
print("\nCorrelation Matrix:\n", df.corr())
print("\nSummary Statistics:\n", df.describe())
```

```
Various Statistical Measures

Mean of columns:
 A    30.0
B    25.0
C     6.0
dtype: float64

Median of columns:
 A    30.0
B    25.0
C     6.0
dtype: float64

Standard Deviation of columns:
 A    15.811388
B    15.811388
C     3.162278
dtype: float64

Variance of columns:
 A    250.0
B    250.0
C     10.0
dtype: float64

Minimum of columns:
 A    10
B     5
C     2
dtype: int64

Maximum of columns:
 A    50
B    45
C    10
dtype: int64

Correlation Matrix:
      A    B    C
A  1.0  1.0  1.0
B  1.0  1.0  1.0
C  1.0  1.0  1.0
```

```
Summary Statistics:
               A          B          C
count   5.000000   5.000000   5.000000
mean   30.000000  25.000000   6.000000
std    15.811388  15.811388   3.162278
min    10.000000   5.000000   2.000000
25%    20.000000  15.000000   4.000000
50%    30.000000  25.000000   6.000000
75%    40.000000  35.000000   8.000000
max    50.000000  45.000000  10.000000
```

1.24 WAP to perform read and write operations with csv files.

[137]:
```python
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}
df = pd.DataFrame(data)
```

[138]:
```python
# Writing to CSV
df.to_csv('sample.csv', index=False)
print("Data written to 'sample.csv'")
```

```
Data written to 'sample.csv'
```

[140]:
```python
# Reading from CSV
df_read = pd.read_csv('sample.csv')
print("\nData read from 'sample.csv':")
df_read
```

```
Data read from 'sample.csv':
```

[140]:

	Name	Age	City
0	Alice	25	New York
1	Bob	30	Los Angeles
2	Charlie	35	Chicago
3	David	40	Houston

1.25 WAP to compute values of $\sin(x)$ using taylor series.

[129]:
```python
import math

def taylor_sin(x, terms=10):
    """
    Computes sin(x) using Taylor series expansion.
```

```
    Parameters:
        x (float): The angle in radians.
        terms (int): Number of terms in the Taylor series.

    Returns:
        float: Approximation of sin(x).
    """
    sin_x = 0
    for n in range(terms):
        term = ((-1)**n) * (x**(2*n + 1)) / math.factorial(2*n + 1)
        sin_x += term
    return sin_x

angle_deg = float(input("Enter the angle in degrees: "))
angle_rad = math.radians(angle_deg)

approx_sin = taylor_sin(angle_rad, terms=10)
exact_sin = math.sin(angle_rad)

# Output
print(f"Using Taylor Series, sin({angle_deg}°)   {approx_sin}")
print(f"Exact value from math.sin(), sin({angle_deg}°) = {exact_sin}")
```

```
Using Taylor Series, sin(30.0°)   0.49999999999999994
Exact value from math.sin(), sin(30.0°) = 0.49999999999999994
```

1.26 WAP to display the following pattern

```
    5
   45
  345
 2345
12345
```

[75]:
```
n = int(input("Enter N:"))
for i in range(n,0,-1):
    print('  '*(i-1),end='')
    for j in range(i,n+1):
        print(j,end=' ')
    print()
```

```
        5
      4 5
    3 4 5
  2 3 4 5
1 2 3 4 5
```

1.27 WAP to find if a number or string is palindrome or not.

[102]:
```
s = input("Enter a number or a string: ")

def isPalindrome(s):
    return s == s[::-1]

print(f"{s} is {"" if isPalindrome(s) else "NOT "}a palindrome.")
```

```
232 is a palindrome.
```

1.28 WAP to find greatest of number using loop.

[29]:
```
def greatest(l:list)->int:
    grt = l[0]
    for i in range(1,len(l)):
        if l[i]>grt:
            grt = l[i]
    return grt
```

[35]:
```
# sample run

l = [1, 2, 3, 1, 4, -5, 5, 0, -1, 1]

print("Greatest number in the given list:",greatest(l))
```

```
Greatest number in the given list: 5
```

1.29 WAP to print fibonacci series.

[96]:
```
def fib(n):
    """Returns nth fibonacci element"""
    if n<=2:
        return 1
    return fib(n-1)+fib(n-2)
def fib_series(n):
    """Returns fibonacci series till `n` elements."""
    return [fib(x) for x in range(1,n+1)]

n = int(input("Enter length of fibonacci series to be printed: ")) # 10
print(fib_series(n))
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

1.30 WAP to find factorial using recursion.

[12]:
```
def fac(n):
    return 1 if n==0 else n*fac(n-1)
```

[13]:
```
# Sample Run

n = int(input("Enter a number: "))
assert n>=0
print(f"{n}! = {fac(n)}")
```

```
5! = 120
```

1.31 WAP to find if a number is armstrong or not.

Armstrong Number: An integer such that the sum of the cubes of its digits is equal to the number itself.

[17]:
```
def is_armstrong(n):
    return n == sum([(int(x))**3 for x in list(str(n))])
```

[18]:
```
# Sample Run 1
n = int(input("Enter a number to check if it is Armstrong or not: "))

print(f"{n} is {"" if is_armstrong(n) else "NOT "}an Armstrong Number.")

# Sample Run 2
n = int(input("Enter a number to check if it is Armstrong or not: "))

print(f"{n} is {"" if is_armstrong(n) else "NOT "}an Armstrong Number.")
```

```
153 is an Armstrong Number.
379 is NOT an Armstrong Number.
```

1.32 Write a menu driven program to find the reverse of a number and sum of digits.

[9]:
```
def reverse(n):
    return int(str(n)[::-1])

def sum_of_digits(n):
    return sum([int(x) for x in list(str(n))])

while True:
    c = input("1) Reverse Number.\n2) Sum of digits.\n3) Exit. \nEnter your␣
↪choice (1-3):")
    if c == "3":
        print("Exit Successful")
        break
```

```python
    elif c == "1":
        n = int(input("Enter the Number: "))
        print(f"Reverse of {n} is {reverse(n)}")
    elif c == "2":
        n = int(input("Enter the Number: "))
        print(f"Sum of digits of {n} is {sum_of_digits(n)}")
    else:
        print("ERROR: Please select a valid choice (1-3).")
```

```
Reverse of 123 is 321
Sum of digits of 123 is 6
ERROR: Please select a valid choice (1-3).
Exit Successful
```
