Rayat Shikshan Sanstha's

KARMAVEER BHAURAO PATIL COLLEGE, VASHI

[Autonomous College]

Reaccredited by NAAC with Grade 'A+' (CGPA 3.53) | ISO 9001: 2008 Certified Institute

'Best College' Award by University of Mumbai

Optimization Methods For Data Science

NAME : Shahzaad Khan

ROLL NO : 240726

CLASS : Msc Part 2(Data Science)

SUBJECT : Natural Language Processing

"Education through self-help is our Motto"-KARMAVEER

RAYAT SHIKSHAN SANSTHA'S

Karmaveer Bhaurao Patil College

Vashi, Navi Mumbai – 400703

(Empowered Autonomous College)

DEPARTMENT OF DATA SCIENCE

CERTIFICATE

This is to certify that **Mr.Shahzaad Firoz Khan** student of M.Sc. Part-II Data Science class from Karmaveer Bhaurao Patil College, Vashi, Navi Mumbai has satisfactorily completed the Practical cource in **Optimization Methods For Data Science** during the academic year 2024-2025.

Roll No: PG - 240726

Exam No:

In charge Faculty
Date : / /2025

Examine

Head
Department of Data Science

Index

Sr.No	Practicals Name	Date	Signature
1	Create a simple mathematical model for a real-world problem (e.g., maximizing profit for a small business) using Python's PuLP library or R's lpSolve package.		
2	Analyze a case study that demonstrates the scope and application of Operations Research in industries like healthcare, logistics, or finance. Present findings using data visualization (e.g.,matplotlib in Python or ggplot2 in R).		
3	Formulate and solve a linear programming problem using the graphical method. Plot the constraints and feasible region, and identify the optimal solution.		
4	Use Python's SciPy library or R's lpSolve to solve a linear programming problem. Compare results with graphical analysis.		
5	Define a primal linear programming problem and derive its dual. Use a programming language to solve both problems and compare the results.		
6	Implement the Dual Simplex method in either R or Python and analyze its performance on a specific example, discussing the economic interpretation of duality.		
7	Set up a transportation problem using Python's PuLP or R's transport package. Solve a balanced transportation problem and visualize the flow using a network graph.		
8	Solve an assignment problem using the Hungarian method. Implement this using Python's scipy.optimize.linear_sum_assignment function or R's clue package.		
9	Construct a project network using CPM (Critical Path Method) in Python or R. Calculate the critical path and project duration.		
10	Simulate a PERT (Program Evaluation and Review Technique) analysis by generating random durations for tasks. Calculate expected project completion time and variance.		

Practical No:-1

Aim:- Create a simple mathematical model for a real-world problem (e.g., maximizing profit for a small business) using Python's PuLP library or R's lpSolve package.

Example 2.2. A retail store stocks two types of shirts *A* and *B*. These are packed in attractive cardboard boxes. During a week the store can sell a maximum of 400 shirts of type *A* and a maximum of 300 shirts of type *B*. The storage capacity, however, is limited to a maximum of 600 of both types combined. Type *A* shirt fetches a profit of Rs. 2/- per unit and type *B* a profit of Rs. 5/- per unit. How many of each type the store should stock per week to maximize the total profit? Formulate a mathematical model of the problem.

1. **Formulate the Problem:**
    - Let x be the number of type A shirts.
    - Let y be the number of type B shirts.
    - The profit from type A shirts is Rs. 2 per unit.
    - The profit from type B shirts is Rs. 5 per unit.
    - The constraints are:
        - $x \leq 400$ (maximum sales of type A)
        - $y \leq 300$ (maximum sales of type B)
        - $x+y \leq 600$ (storage capacity)
        - $x \geq 0$, $y \geq 0$ (non-negativity)

Code:-

```
! pip install pulp
import pulp
prob = pulp.LpProblem("Maximize_Profit", pulp.LpMaximize)
x = pulp.LpVariable('x', lowBound=0, cat='Integer')  # Number of type A shirts
y = pulp.LpVariable('y', lowBound=0, cat='Integer')  # Number of type B shirts
prob += 2*x + 5*y, "Total_Profit"
prob += x <= 400, "Max_Sales_A"
prob += y <= 300, "Max_Sales_B"
prob += x + y <= 600, "Storage_Capacity"
prob.solve()
print("Status:", pulp.LpStatus[prob.status])
print("Number of type A shirts to stock:", pulp.value(x))
```

print("Number of type B shirts to stock:", pulp.value(y))

print("Total Profit:", pulp.value(prob.objective))

Output:-

```
Collecting pulp
  Downloading PuLP-3.0.2-py3-none-any.whl.metadata (6.7 kB)
Downloading PuLP-3.0.2-py3-none-any.whl (17.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.7/17.7 MB 14.8 MB/s eta 0:00:00
Installing collected packages: pulp
Successfully installed pulp-3.0.2
Status: Optimal
Number of type A shirts to stock: 300.0
Number of type B shirts to stock: 300.0
Total Profit: 2100.0
```

Practical No:-2

Aim:- Analyze a case study that demonstrates the scope and application of Operations Research in industries like healthcare, logistics, or finance. Present findings using data visualization (e.g., matplotlib in Python or ggplot2 in R).

Case Study: Optimizing Delivery Routes in Logistics

Problem Statement:
A delivery company needs to optimize its delivery routes to minimize fuel costs and delivery time. The company has a fleet of vehicles and a set of delivery locations. Each vehicle has a limited capacity, and each delivery location has a specific demand.

Data:

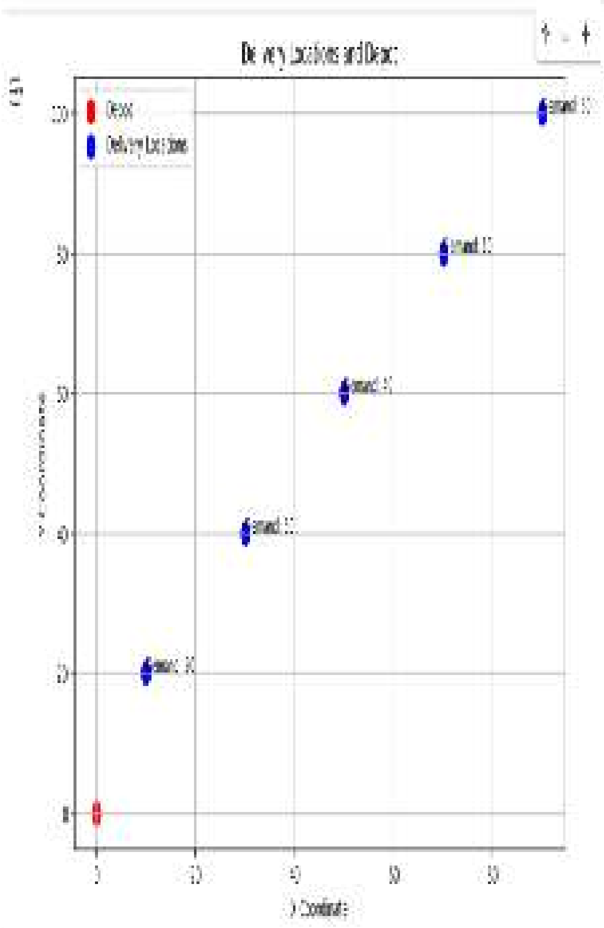
- Delivery Locations: Coordinates (x, y) and demand.
- Vehicle Capacity: Each vehicle can carry up to 100 units.
- Depot Location: Central warehouse where all vehicles start and end their routes.

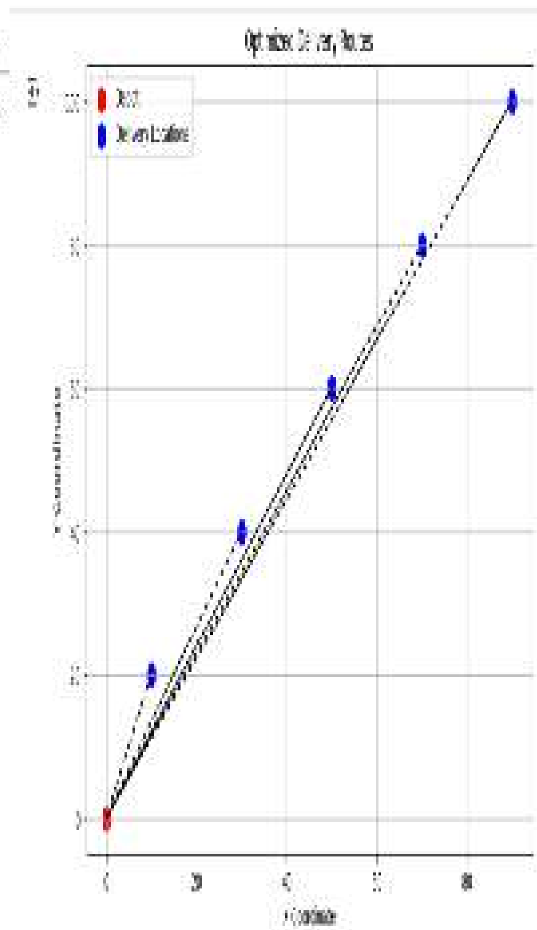
Code:-

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.spatial.distance import cdist

depot = np.array([[0, 0]])

locations = np.array([[10, 20], [30, 40], [50, 60], [70, 80], [90, 100]])

demands = np.array([20, 30, 40, 10, 50])

distances = cdist(depot, locations)

plt.figure(figsize=(10, 6))

plt.scatter(depot[:, 0], depot[:, 1], c='red', label='Depot', s=100) # Removed extra ')'

plt.scatter(locations[:, 0], locations[:, 1], c='blue', label='Delivery Locations', s=100)

for i, (x, y) in enumerate(locations):

   plt.text(x, y, f'Demand: {demands[i]}', fontsize=9)

plt.title('Delivery Locations and Depot')

plt.xlabel('X Coordinate')

plt.ylabel('Y Coordinate')

plt.legend()

plt.grid(True)
```

```
plt.show()
plt.figure(figsize=(10, 6))
plt.scatter(depot[:, 0], depot[:, 1], c='red', label='Depot', s=100)
plt.scatter(locations[:, 0], locations[:, 1], c='blue', label='Delivery Locations', s=100)
for i, (x, y) in enumerate(locations):
    plt.plot([depot[0, 0], x], [depot[0, 1], y], 'k--')
plt.title('Optimized Delivery Routes')
plt.xlabel('X Coordinate')
plt.ylabel('Y Coordinate')
plt.legend()
plt.grid(True)
plt.show()
```

Output:-





Practical No:-3

Aim:- Formulate and solve a linear programming problem using the graphical method. Plot the constraints and feasible region, and identify the optimal solution.

Problem 2.6. A company manufactures two products, *X* and *Y* by using three machines *A*, *B*, and *C*. Machine *A* has 4 hours of capacity available during the coming week. Similarly, the available capacity of machines *B* and *C* during the coming week is 24 hours and 35 hours respectively. One unit of

product *X* requires one hour of Machine *A*, 3 hours of machine *B* and 10 hours of machine *C*. Similarly one unit of product *Y* requires 1 hour, 8 hour and 7 hours of machine *A*, *B* and *C* respectively. When one unit of *X* is sold in the market, it yields a profit of Rs. 5/- per product and that of *Y* is Rs. 7/- per unit. Solve the problem by using graphical method to find the optimal product mix.

Problem Statement:

Maximize Z=5x+7y

1. **Formulate the Constraints**:
    - Machine A constraint: $1x+1y\leq4$
    - Machine B constraint: $3x+8y\leq243$
    - Machine C constraint: $10x+7y\leq35$
    - Non-negativity constraints: $x\geq0$, $y\geq0$

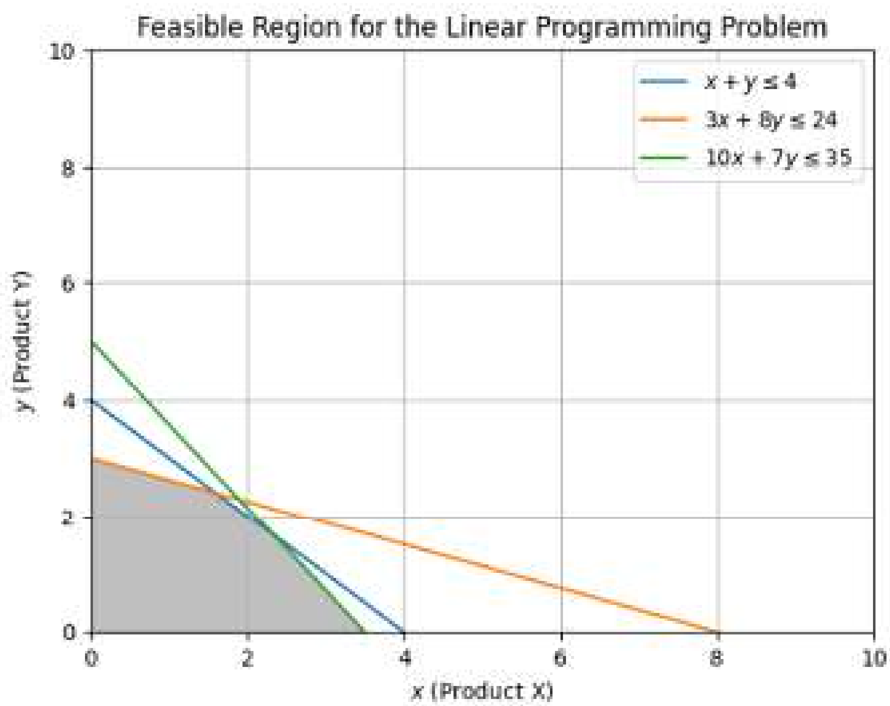
Code:-

```
import numpy as np

import matplotlib.pyplot as plt

x = np.linspace(0, 10, 400)

y1 = 4 - x

y2 = (24 - 3*x) / 8

y3 = (35 - 10*x) / 7

plt.plot(x, y1, label=r'$x + y \leq 4$')

plt.plot(x, y2, label=r'$3x + 8y \leq 24$')

plt.plot(x, y3, label=r'$10x + 7y \leq 35$')

y4 = np.minimum.reduce([y1, y2, y3])

plt.fill_between(x, 0, y4, where=(y4 >= 0), color='gray', alpha=0.5)

plt.xlim((0, 10))

plt.ylim((0, 10))
```

```
plt.xlabel(r'$x$ (Product X)')
plt.ylabel(r'$y$ (Product Y)')
plt.legend()
plt.title('Feasible Region for the Linear Programming Problem')
plt.grid(True)
plt.show()
A1 = np.array([[1, 1], [3, 8]])
b1 = np.array([4, 24])
intersection1 = np.linalg.solve(A1, b1)
A2 = np.array([[1, 1], [10, 7]])
b2 = np.array([4, 35])
intersection2 = np.linalg.solve(A2, b2)
A3 = np.array([[3, 8], [10, 7]])
b3 = np.array([24, 35])
intersection3 = np.linalg.solve(A3, b3)
x_axis_intersection1 = 4  # From x + 0 = 4
x_axis_intersection2 = 8  # From 3x + 0 = 24 => x = 8
x_axis_intersection3 = 3.5  # From 10x + 0 = 35 => x = 3.5
y_axis_intersection1 = 4  # From 0 + y = 4
y_axis_intersection2 = 3  # From 0 + 8y = 24 => y = 3
y_axis_intersection3 = 5  # From 0 + 7y = 35 => y = 5
print(f"Intersection of x + y = 4 and 3x + 8y = 24: x = {intersection1[0]}, y = {intersection1[1]}")
print(f"Intersection of x + y = 4 and 10x + 7y = 35: x = {intersection2[0]}, y = {intersection2[1]}")
print(f"Intersection of 3x + 8y = 24 and 10x + 7y = 35: x = {intersection3[0]}, y = {intersection3[1]}")
print(f"Intersection with x-axis (y=0): x = {min(x_axis_intersection1, x_axis_intersection2,
x_axis_intersection3)}")
print(f"Intersection with y-axis (x=0): y = {min(y_axis_intersection1, y_axis_intersection2,
y_axis_intersection3)}")
Z1 = 5 * intersection1[0] + 7 * intersection1[1]
Z2 = 5 * intersection2[0] + 7 * intersection2[1]
Z3 = 5 * intersection3[0] + 7 * intersection3[1]
```

```
Z4 = 5 * min(x_axis_intersection1, x_axis_intersection2, x_axis_intersection3) + 7 * 0
Z5 = 5 * 0 + 7 * min(y_axis_intersection1, y_axis_intersection2, y_axis_intersection3)
print(f"Objective function value at intersection1: Z = {Z1}")
print(f"Objective function value at intersection2: Z = {Z2}")
print(f"Objective function value at intersection3: Z = {Z3}")
print(f"Objective function value at x-axis intersection: Z = {Z4}")
print(f"Objective function value at y-axis intersection: Z = {Z5}")
max_Z = max(Z1, Z2, Z3, Z4, Z5)
print(f"Maximum profit Z = {max_Z}")
```

Output:-




```
Intersection of x + y = 4 and 3x + 8y = 24: x = 1.5999999999999999, y = 2.4000000000000004
Intersection of x + y = 4 and 10x + 7y = 35: x = 2.3333333333333333, y = 1.6666666666666667
Intersection of 3x + 8y = 24 and 10x + 7y = 35: x = 1.8983050847457625, y = 2.288135593220339
Intersection with x-axis (y=0): x = 3.5
Intersection with y-axis (x=0): y = 3
Objective function value at intersection1: Z = 24.799999999999997
Objective function value at intersection2: Z = 23.333333333333332
Objective function value at intersection3: Z = 25.508474576271187
Objective function value at x-axis intersection: Z = 17.5
Objective function value at y-axis intersection: Z = 21
Maximum profit Z = 25.508474576271187
```

Practical No:-4

Aim:- Use Python's SciPy library or R's lpSolve to solve a linear programming problem. Compare results with graphical analysis.

Problem 2.7. Solve graphically the given linear programming problem.

Minimize $Z = 3a + 5b$ S.T

$-3a + 4b \le 12$

$2a - 1b \ge -2$

$2a + 3b \ge 12$

$1a + 0b \ge 4$

$0a + 1b \ge 2$

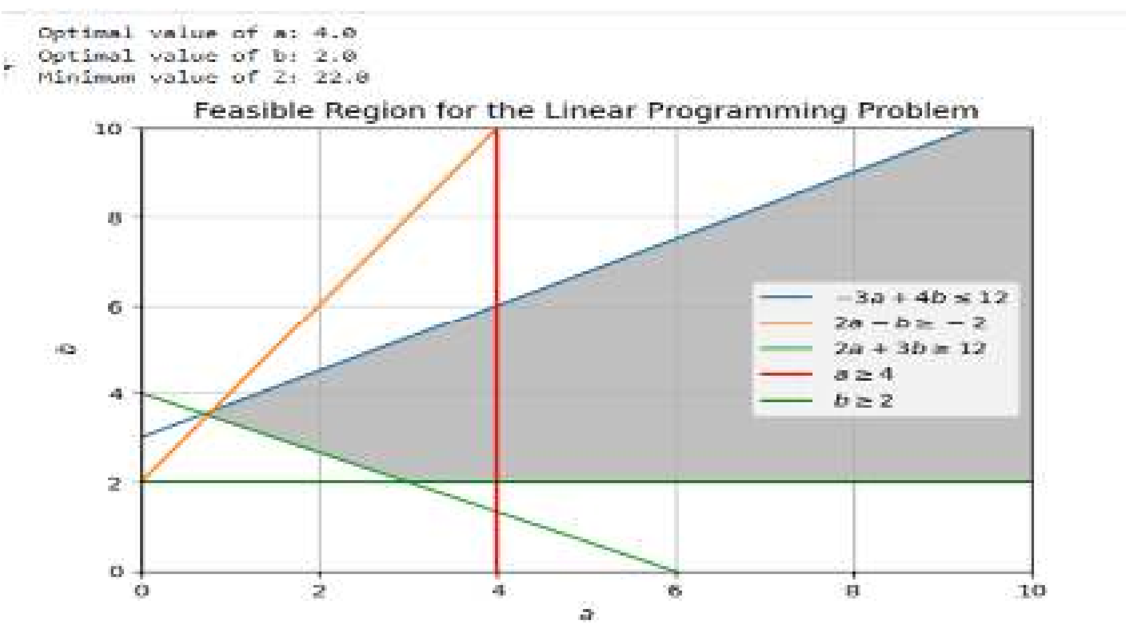
And both a and b are ≥ 0.

Code:-

```
from scipy.optimize import linprog
c = [3, 5]  # Coefficients for a and b in Z = 3a + 5b
A = [
    [-3, 4],   # -3a + 4b <= 12
    [-2, 1],   # 2a - b >= -2 => -2a + b <= 2
    [-2, -3],  # 2a + 3b >= 12 => -2a - 3b <= -12
    [-1, 0],   # a >= 4 => -a <= -4
    [0, -1]    # b >= 2 => -b <= -2
]
b = [12, 2, -12, -4, -2]
a_bounds = (4, None)  # a >= 4
b_bounds = (2, None)  # b >= 2
result = linprog(c, A_ub=A, b_ub=b, bounds=[a_bounds, b_bounds], method='highs')
print("Optimal value of a:", result.x[0])
print("Optimal value of b:", result.x[1])
print("Minimum value of Z:", result.fun)
import numpy as np
import matplotlib.pyplot as plt


a = np.linspace(0, 10, 400)
b1 = (12 + 3*a) / 4  # -3a + 4b <= 12
```

```
b2 = 2 + 2*a         # 2a - b >= -2 => b <= 2 + 2a
b3 = (12 - 2*a) / 3   # 2a + 3b >= 12 => b >= (12 - 2a) / 3
plt.plot(a, b1, label=r'$-3a + 4b \leq 12$')
plt.plot(a, b2, label=r'$2a - b \geq -2$')
plt.plot(a, b3, label=r'$2a + 3b \geq 12$')
plt.axvline(x=4, color='r', label=r'$a \geq 4$')
plt.axhline(y=2, color='g', label=r'$b \geq 2$')
b4 = np.maximum.reduce([b3, np.full_like(a, 2)])
b5 = np.minimum.reduce([b1, b2])
plt.fill_between(a, b4, b5, where=(b4 <= b5), color='gray', alpha=0.5)
plt.xlim((0, 10))
plt.ylim((0, 10))
plt.xlabel(r'$a$')
plt.ylabel(r'$b$')
plt.legend()
plt.title('Feasible Region for the Linear Programming Problem')
plt.grid(True)
plt.show()
```

Output:-



Practical No:-5

Aim:- Define a primal linear programming problem and derive its dual. Use a programming language to solve both problems and compare the results.

Problem 3.32: A company manufactures two products *X* and *Y* on three machines Turning, Milling and finishing machines. Each unit of *X* takes, 10 hours of turning machine capacity, 5 hours of milling machine capacity and 1 hour of finishing machine capacity. One unit of *Y* takes 6 hours of turning machine capacity, 10 hours of milling machine capacity and 2 hours of finishing machine capacity. The company has 2500 hours of turning machine capacity, 2000 hours of milling machine capacity and 500 hours of finishing machine capacity in the coming planning period. The profit contribution of product *X* and *Y* are Rs. 23 per unit and Rs. 32 per unit respectively. Formulate the linear programming problems and write the dual.

Primal Problem:

Decision Variables:

- Let x be the number of units of product X.
- Let y be the number of units of product Y.

Objective Function:

- Maximize profit: $Z=23x+32y$.

Constraints:

1. Turning machine capacity: $10x+6y \leq 2500$.
2. Milling machine capacity: $5x+10y \leq 2000$.
3. Finishing machine capacity: $x+2y \leq 500$.
4. Non-negativity: $x \geq 0$, $y \geq 0$.

Dual Problem Formulation

The dual problem is derived from the primal problem. For a primal problem in the standard form:

Primal:

- Maximize $Z=cTx$
- Subject to $Ax \leq b$, $x \geq 0$

Dual:

- Minimize $W=bTy$
- Subject to $ATy \geq c$, $y \geq 0$

For our problem:

Decision Variables:

- Let $y1$ be the dual variable for the turning machine constraint.

- Let y2 be the dual variable for the milling machine constraint.
- Let y3 be the dual variable for the finishing machine constraint.

Objective Function:

- Minimize W=2500y1+2000y2+500y3.

Constraints:

1. 10y1+5y2+y3≥23.
2. 6y1+10y2+2y3≥32.
3. Non-negativity: y1≥0, y2≥0, y3≥0.

Code:-

```
from scipy.optimize import linprog

c_primal = [-23, -32]  # Coefficients for x and y in Z = 23x + 32y (minimize -Z)

A_primal = [

    [10, 6],   # Turning machine

    [5, 10],   # Milling machine

    [1, 2]     # Finishing machine

]

b_primal = [2500, 2000, 500]

bounds_primal = [(0, None), (0, None)]  # x >= 0, y >= 0

result_primal = linprog(c_primal, A_ub=A_primal, b_ub=b_primal, bounds=bounds_primal,
method='highs')

c_dual = [2500, 2000, 500]  # Coefficients for y1, y2, y3 in W = 2500y1 + 2000y2 + 500y3

A_dual = [

    [-10, -5, -1],  # 10y1 + 5y2 + y3 >= 23

    [-6, -10, -2]   # 6y1 + 10y2 + 2y3 >= 32

]

b_dual = [-23, -32]

bounds_dual = [(0, None), (0, None), (0, None)]  # y1 >= 0, y2 >= 0, y3 >= 0

result_dual = linprog(c_dual, A_ub=A_dual, b_ub=b_dual, bounds=bounds_dual, method='highs')
```

```
print("Primal Problem:")
print("Optimal value of x:", result_primal.x[0])
print("Optimal value of y:", result_primal.x[1])
print("Maximum profit Z:", -result_primal.fun)
print("\nDual Problem:")
print("Optimal value of y1:", result_dual.x[0])
print("Optimal value of y2:", result_dual.x[1])
print("Optimal value of y3:", result_dual.x[2])
print("Minimum cost W:", result_dual.fun)
```

Output:-

```
Primal Problem:
Optimal value of x: 185.71428571428572
Optimal value of y: 107.14285714285711
Maximum profit Z: 7699.999999999999

Dual Problem:
Optimal value of y1: 1.0
Optimal value of y2: 2.6
Optimal value of y3: 0.0
Minimum cost W: 7700.0
```

Practical No:-6

Aim:- Implement the Dual Simplex method in either R or Python and analyze its performance on a specific example, discussing the economic interpretation of duality.

Problem 3.34: Construct the dual of the given l.p.p.
Maximize $Z = 5w + 2x + 6y + 4z$ s.t
$1w + 1x + 1y + 1z \leq 140$
$2w + 5x + 6y + 1z \geq 200$
$1w + 3x + 1y + 2z \leq 150$
And w, x, y, z all are ≥ 0.

Problem Statement

Primal Problem:
Maximize $Z=5w+2x+6y+4z$

subject to:

1. $w+x+y+z \leq 140$
2. $w+5x+6y+z \geq 200$
3. $w+3x+y+2z \leq 150$
4. $w,x,y,z \geq 0$

Dual Problem Formulation

The dual problem is derived from the primal problem. For a primal problem in the standard form:

Primal:

- Maximize Z=cTx
- Subject to Ax≤b, x≥0

Dual:

- Minimize W=bTy
- Subject to ATy≥c, y≥0

Decision Variables:

- Let y1 be the dual variable for the first constraint.
- Let y2 be the dual variable for the second constraint.
- Let y3 be the dual variable for the third constraint.

Objective Function:

- Minimize $W=140y1+200y2+150y3$.

Constraints:

1. y1+y2+y3≥5.
2. y1+5y2+3y3≥2.
3. y1+6y2+y3≥6.
4. y1+y2+2y3≥4.
5. Non-negativity: y1≥0, y2≥0, y3≥0.

Code:-

```
from scipy.optimize import linprog

c_primal = [-5, -2, -6, -4]  # Coefficients for w, x, y, z in Z = 5w + 2x + 6y + 4z (minimize -Z)

A_primal = [

    [1, 1, 1, 1],   # w + x + y + z <= 140

    [-1, -5, -6, -1],  # w + 5x + 6y + z >= 200 => -w - 5x - 6y - z <= -200

    [1, 3, 1, 2]    # w + 3x + y + 2z <= 150

]

b_primal = [140, -200, 150]

bounds_primal = [(0, None), (0, None), (0, None), (0, None)]  # w, x, y, z >= 0

result_primal = linprog(c_primal, A_ub=A_primal, b_ub=b_primal, bounds=bounds_primal,
method='revised simplex')

c_dual = [140, 200, 150]  # Coefficients for y1, y2, y3 in W = 140y1 + 200y2 + 150y3

A_dual = [

    [-1, -1, -1],  # y1 + y2 + y3 >= 5

    [-1, -5, -3],  # y1 + 5y2 + 3y3 >= 2

    [-1, -6, -1],  # y1 + 6y2 + y3 >= 6

    [-1, -1, -2]   # y1 + y2 + 2y3 >= 4

]

b_dual = [-5, -2, -6, -4]

bounds_dual = [(0, None), (0, None), (0, None)]  # y1 >= 0, y2 >= 0, y3 >= 0

result_dual = linprog(c_dual, A_ub=A_dual, b_ub=b_dual, bounds=bounds_dual, method='highs')

print("Primal Problem:")

print("Optimal value of w:", result_primal.x[0])

print("Optimal value of x:", result_primal.x[1])
```

```
print("Optimal value of y:", result_primal.x[2])
print("Optimal value of z:", result_primal.x[3])
print("Maximum profit Z:", -result_primal.fun)
print("\nDual Problem:")
print("Optimal value of y1:", result_dual.x[0])
print("Optimal value of y2:", result_dual.x[1])
print("Optimal value of y3:", result_dual.x[2])
print("Minimum cost W:", result_dual.fun)
```

Output:-

```
Primal Problem:
Optimal value of w: 0.0
Optimal value of x: 0.0
Optimal value of y: 140.0
Optimal value of z: 0.0
Maximum profit Z: 840.0

Dual Problem:
Optimal value of y1: 4.8
Optimal value of y2: 0.2
Optimal value of y3: 0.0
Minimum cost W: 712.0
<ipython-input-7-6a2ffcf0c1e0>:14: DeprecationWarning: `method='revised simplex'` is
  result_primal = linprog(c_primal, A_ub=A_primal, b_ub=b_primal, bounds=bounds_prima
```

Practical No:-7

Aim:- Set up a transportation problem using Python's PuLP or R's transport package. Solve a balanced transportation problem and visualize the flow using a network graph.

Example 4.1. Four factories, *A*, *B*, *C* and *D* produce sugar and the capacity of each factory is given below: Factory *A* produces 10 tons of sugar and *B* produces 8 tons of sugar, *C* produces 5 tons of sugar and that of *D* is 6 tons of sugar. The sugar has demand in three markets *X*, *Y* and *Z*. The demand of market *X* is 7 tons, that of market *Y* is 12 tons and the demand of market *Z* is 4 tons. The following matrix gives the transportation cost of 1 ton of sugar from each factory to the destinations. Find the Optimal Solution for least cost transportation cost.

Code:-

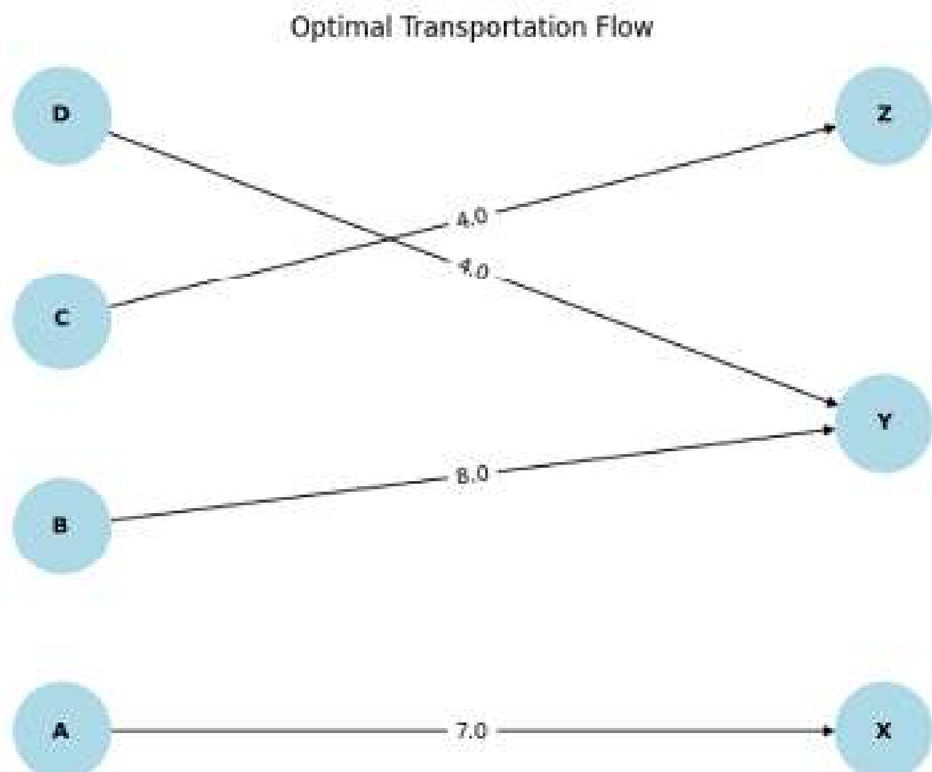
```
import pulp

import networkx as nx

import matplotlib.pyplot as plt

supply = {

    'A': 10,

    'B': 8,

    'C': 5,

    'D': 6

}

demand = {

    'X': 7,

    'Y': 12,

    'Z': 4

}

costs = {

    ('A', 'X'): 2,

    ('A', 'Y'): 4,

    ('A', 'Z'): 5,

    ('B', 'X'): 3,

    ('B', 'Y'): 1,

    ('B', 'Z'): 6,

    ('C', 'X'): 7,
```

```
    ('C', 'Y'): 3,
    ('C', 'Z'): 2,
    ('D', 'X'): 4,
    ('D', 'Y'): 2,
    ('D', 'Z'): 3
}
prob = pulp.LpProblem("Transportation_Problem", pulp.LpMinimize)
routes = [(f, m) for f in supply for m in demand]
x = pulp.LpVariable.dicts("Route", (supply, demand), lowBound=0, cat='Integer')
prob += pulp.lpSum([x[f][m] * costs[(f, m)] for (f, m) in routes]), "Total_Transportation_Cost"
for f in supply:
    prob += pulp.lpSum([x[f][m] for m in demand]) <= supply[f], f"Supply_{f}"
for m in demand:
    prob += pulp.lpSum([x[f][m] for f in supply]) >= demand[m], f"Demand_{m}"
prob.solve()
print("Status:", pulp.LpStatus[prob.status])
print("Total Transportation Cost:", pulp.value(prob.objective))
for f in supply:
    for m in demand:
        print(f"Quantity transported from {f} to {m}: {x[f][m].varValue}")
G = nx.DiGraph()
G.add_nodes_from(supply.keys(), bipartite=0)
G.add_nodes_from(demand.keys(), bipartite=1)
for f in supply:
    for m in demand:
        if x[f][m].varValue > 0:
            G.add_edge(f, m, weight=x[f][m].varValue)
pos = nx.bipartite_layout(G, supply.keys())
nx.draw(G, pos, with_labels=True, node_color='lightblue', node_size=2000, font_size=10,
font_weight='bold')
```

```
labels = nx.get_edge_attributes(G, 'weight')

nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)

plt.title("Optimal Transportation Flow")

plt.show()
```

Output:-

```
Status: Optimal
Total Transportation Cost: 38.0
Quantity transported from A to X: 7.0
Quantity transported from A to Y: 0.0
Quantity transported from A to Z: 0.0
Quantity transported from B to X: 0.0
Quantity transported from B to Y: 8.0
Quantity transported from B to Z: 0.0
Quantity transported from C to X: 0.0
Quantity transported from C to Y: 0.0
Quantity transported from C to Z: 4.0
Quantity transported from D to X: 0.0
Quantity transported from D to Y: 4.0
Quantity transported from D to Z: 0.0
```



Practical No:-8

Aim:- Solve an assignment problem using the Hungarian method. Implement this using Python's scipy.optimize.linear_sum_assignment function or R's clue package.

Problem 5.3.

Five jobs are to be assigned to 5 machines to minimize the total time required to process the jobs on machines. The times in hours for processing each job on each machine are given in the matrix below. By using assignment algorithm make the assignment for minimizing the time of processing.

Machines (time in hours)

Jobs	*V*	*W*	*X*	*Y*	*Z*
A	2	4	3	5	4
B	7	4	6	8	4
C	2	9	8	10	4
D	8	6	12	7	4
E	2	8	5	8	8

Code:-

```
import numpy as np
from scipy.optimize import linear_sum_assignment
cost_matrix = np.array([
    [2, 4, 3, 5, 4],
    [7, 4, 6, 8, 4],
    [2, 9, 8, 10, 4],
    [8, 6, 12, 7, 4],
    [2, 8, 5, 8, 8] ])
row_ind, col_ind = linear_sum_assignment(cost_matrix)
total_time = cost_matrix[row_ind, col_ind].sum()
print("Optimal assignment:")
for i, j in zip(row_ind, col_ind):
    print(f"Machine {chr(65 + i)} is assigned to Job {chr(86 + j)} with time {cost_matrix[i, j]} hours")
print(f"\nTotal minimum processing time: {total_time} hours")
```

Output:-

```
Optimal assignment:
Machine A is assigned to Job X with time 3 hours
Machine B is assigned to Job W with time 4 hours
Machine C is assigned to Job Z with time 4 hours
Machine D is assigned to Job Y with time 7 hours
Machine E is assigned to Job V with time 2 hours

Total minimum processing time: 20 hours
```

Practical No:-9

Aim:- Construct a project network using CPM (Critical Path Method) in Python or R. Calculate the critical path and project duration.

Problem 15.5.

A company manufacturing plant and equipment for chemical processing is in the process of quoting tender called by public sector undertaking. Help the manager to find the project completion time to participate in the tender.

S. No.	*Activities*		*Days*
1	A	—	3
2	B	—	4
3	C	A	5
4	D	A	6
5	E	C	7
6	F	D	8
7	G	B	9
8	H	E, F, G	3

Code:-

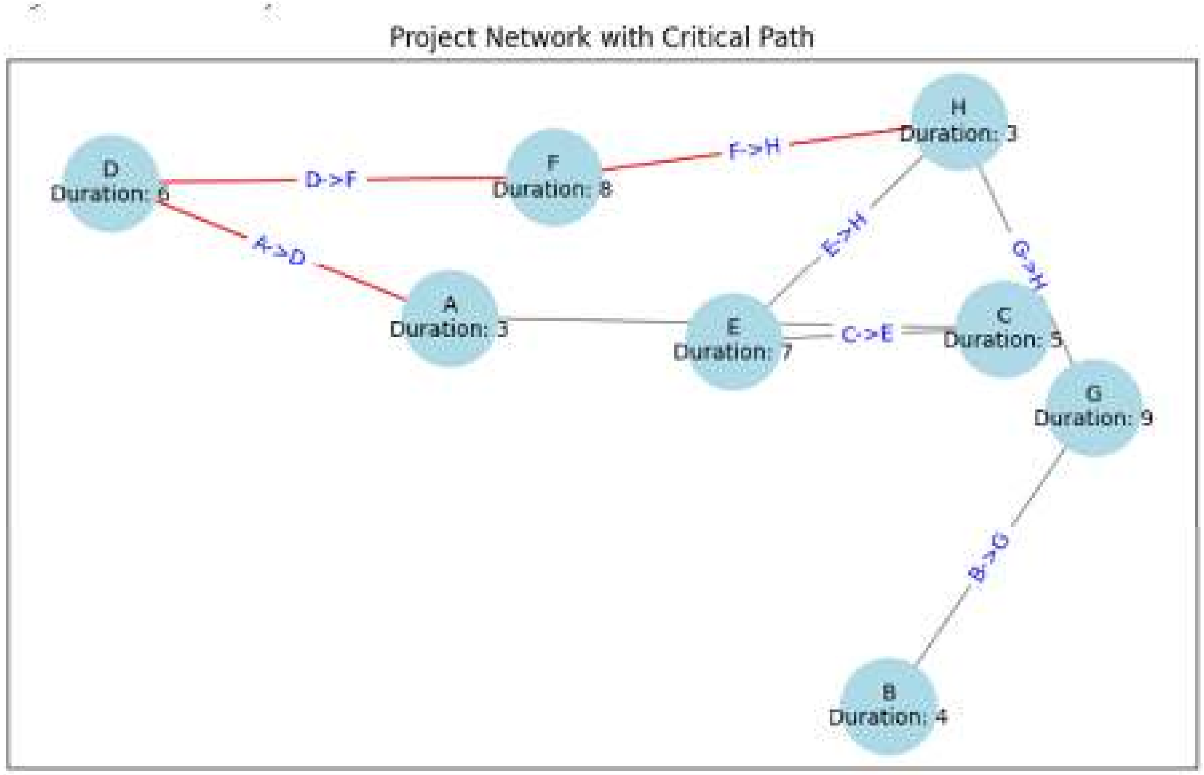
```
import networkx as nx

import matplotlib.pyplot as plt

activities = {

    'A': {'duration': 3, 'dependencies': []},

    'B': {'duration': 4, 'dependencies': []},

    'C': {'duration': 5, 'dependencies': ['A']},

    'D': {'duration': 6, 'dependencies': ['A']},

    'E': {'duration': 7, 'dependencies': ['C']},

    'F': {'duration': 8, 'dependencies': ['D']},

    'G': {'duration': 9, 'dependencies': ['B']},

    'H': {'duration': 3, 'dependencies': ['E', 'F', 'G']}}

G = nx.DiGraph()

for activity, details in activities.items():

    G.add_node(activity, duration=details['duration'])

for activity, details in activities.items():

    for dependency in details['dependencies']:

        G.add_edge(dependency, activity)

earliest_start = {}

earliest_finish = {}
```

```
for node in nx.topological_sort(G):
    duration = G.nodes[node]['duration']
    if not G.in_edges(node):
        earliest_start[node] = 0
    else:
        earliest_start[node] = max(earliest_finish[prev_node] for prev_node, _ in G.in_edges(node))
    earliest_finish[node] = earliest_start[node] + duration
latest_start = {}
latest_finish = {}
for node in reversed(list(nx.topological_sort(G))):
    duration = G.nodes[node]['duration']
    if not G.out_edges(node):
        latest_finish[node] = earliest_finish[node]
    else:
        latest_finish[node] = min(latest_start[next_node] for _, next_node in G.out_edges(node))
    latest_start[node] = latest_finish[node] - duration
slack = {node: latest_start[node] - earliest_start[node] for node in G.nodes}
critical_path = [node for node in G.nodes if slack[node] == 0]
project_duration = max(earliest_finish.values())
print("Earliest Start Times:", earliest_start)
print("Earliest Finish Times:", earliest_finish)
print("Latest Start Times:", latest_start)
print("Latest Finish Times:", latest_finish)
print("Slack Times:", slack)
print("Critical Path:", critical_path)
print("Project Duration:", project_duration, "days")
pos = nx.spring_layout(G)
plt.figure(figsize=(10, 6))
nx.draw_networkx_nodes(G, pos, node_size=2000, node_color='lightblue')
```

```
nx.draw_networkx_edges(G, pos, arrowstyle='->', arrowsize=20, edge_color='gray')

critical_edges = [(critical_path[i], critical_path[i+1]) for i in range(len(critical_path)-1)]

nx.draw_networkx_edges(G, pos, edgelist=critical_edges, edge_color='red', arrowstyle='->',
arrowsize=20)

labels = {node: f"{node}\nDuration: {G.nodes[node]['duration']}" for node in G.nodes}

nx.draw_networkx_labels(G, pos, labels, font_size=10, font_color='black')

edge_labels = {(u, v): f"{u}->{v}" for u, v in G.edges}

nx.draw_networkx_edge_labels(G, pos, edge_labels, font_color='blue')

plt.title("Project Network with Critical Path")

plt.show()
```

Output:-

```
Earliest Start Times: {'A': 0, 'B': 0, 'C': 3, 'D': 3, 'G': 4, 'E': 8, 'F': 9, 'H': 17}
Earliest Finish Times: {'A': 3, 'B': 4, 'C': 8, 'D': 9, 'G': 13, 'E': 15, 'F': 17, 'H': 20}
Latest Start Times: {'H': 17, 'F': 9, 'E': 10, 'G': 8, 'D': 3, 'C': 5, 'B': 4, 'A': 0}
Latest Finish Times: {'H': 20, 'F': 17, 'E': 17, 'G': 17, 'D': 9, 'C': 10, 'B': 8, 'A': 3}
Slack Times: {'A': 0, 'B': 4, 'C': 2, 'D': 0, 'E': 2, 'F': 0, 'G': 4, 'H': 0}
Critical Path: ['A', 'D', 'F', 'H']
Project Duration: 20 days
```



Practical No:-10

Aim:- Simulate a PERT (Program Evaluation and Review Technique) analysis by generating random durations for tasks. Calculate expected project completion time and variance.

Problem 15.3.

A small project is composed of 7 activities whose time estimates are listed below. Activities are being identified by their beginning (i) and ending (j) node numbers.

Activities		*Time in weeks*		
i	j	t_o	t_l	t_p
1	2	1	1	7
1	3	1	4	7
1	4	2	2	8
2	5	1	1	1
3	5	2	5	14
4	6	2	5	8
5	6	3	6	15

1. Draw the network
2. Calculate the expected variances for each
3. Find the expected project completed time

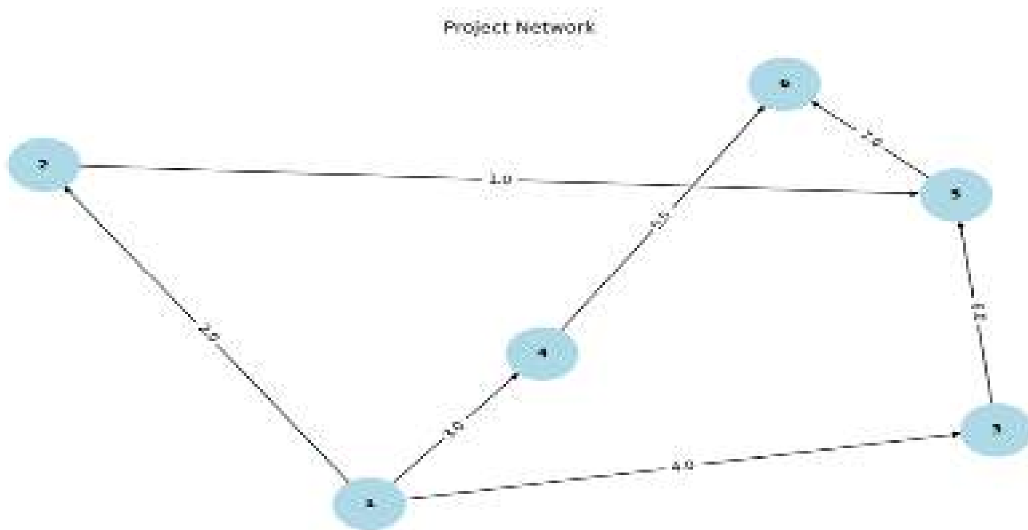
Code:-

```
import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

activities = [

    {'i': 1, 'j': 2, 't_o': 1, 't_m': 1, 't_p': 7},

    {'i': 1, 'j': 3, 't_o': 1, 't_m': 4, 't_p': 7},

    {'i': 1, 'j': 4, 't_o': 2, 't_m': 2, 't_p': 8},

    {'i': 2, 'j': 5, 't_o': 1, 't_m': 1, 't_p': 1},

    {'i': 3, 'j': 5, 't_o': 2, 't_m': 5, 't_p': 14},

    {'i': 4, 'j': 6, 't_o': 2, 't_m': 5, 't_p': 8},

    {'i': 5, 'j': 6, 't_o': 3, 't_m': 6, 't_p': 15}]

for activity in activities:

    t_o, t_m, t_p = activity['t_o'], activity['t_m'], activity['t_p']

    activity['expected_duration'] = (t_o + 4 * t_m + t_p) / 6

    activity['variance'] = ((t_p - t_o) / 6) ** 2

G = nx.DiGraph()
```

```
for activity in activities:
    G.add_edge(activity['i'], activity['j'], weight=activity['expected_duration'])
pos = nx.spring_layout(G)
plt.figure(figsize=(10, 6))
nx.draw(G, pos, with_labels=True, node_color='lightblue', node_size=2000, font_size=10,
font_weight='bold')
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
plt.title("Project Network")
plt.show()
earliest_start = {}
earliest_finish = {}
for node in nx.topological_sort(G):
    if not G.in_edges(node):
        earliest_start[node] = 0
    else:
        earliest_start[node] = max(earliest_finish[prev_node] for prev_node, _ in G.in_edges(node))
    earliest_finish[node] = earliest_start[node] + G.nodes[node].get('weight', 0)
latest_start = {}
latest_finish = {}
for node in reversed(list(nx.topological_sort(G))):
    if not G.out_edges(node):
        latest_finish[node] = earliest_finish[node]
    else:
        latest_finish[node] = min(latest_start[next_node] for _, next_node in G.out_edges(node))
    latest_start[node] = latest_finish[node] - G.nodes[node].get('weight', 0)
slack = {node: latest_start[node] - earliest_start[node] for node in G.nodes}
critical_path = [node for node in G.nodes if slack[node] == 0]
project_completion_time = max(earliest_finish.values())
```

```
project_variance = sum(activity['variance'] for activity in activities if (activity['i'], activity['j']) in critical_edges)

print("Expected Durations and Variances:")

for activity in activities:

    print(f"Activity {activity['i']} -> {activity['j']}: Expected Duration = {activity['expected_duration']:.2f}, Variance = {activity['variance']:.2f}")

print("\nEarliest Start Times:", earliest_start)

print("Earliest Finish Times:", earliest_finish)

print("Latest Start Times:", latest_start)

print("Latest Finish Times:", latest_finish)

print("Slack Times:", slack)

print("Critical Path:", critical_path)

print("Expected Project Completion Time:", project_completion_time, "weeks")

print("Project Variance:", project_variance, "weeks^2")
```

Output:-




```
Expected Durations and Variances:
Activity 1 -> 2: Expected Duration = 2.00, Variance = 1.00
Activity 1 -> 3: Expected Duration = 4.00, Variance = 1.00
Activity 1 -> 4: Expected Duration = 3.00, Variance = 1.00
Activity 2 -> 5: Expected Duration = 1.00, Variance = 0.00
Activity 3 -> 5: Expected Duration = 6.00, Variance = 4.00
Activity 4 -> 6: Expected Duration = 5.00, Variance = 1.00
Activity 5 -> 6: Expected Duration = 7.00, Variance = 4.00

Earliest Start Times: {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0}
Earliest Finish Times: {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0}
Latest Start Times: {6: 0, 5: 0, 4: 0, 3: 0, 2: 0, 1: 0}
Latest Finish Times: {6: 0, 5: 0, 4: 0, 3: 0, 2: 0, 1: 0}
Slack Times: {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0}
Critical Path: [1, 2, 3, 4, 5, 6]
Expected Project Completion Time: 0 weeks
Project Variance: 0 weeks^2
```