DecisionAnalysisPy

Computatonal Tools for Decision Analysis in Python

Kim Leng POH

Department of Industrial Systems Engineering & Management
National University of Singapore
pohkimleng@nus.edu.sg

January 26, 2025

Abstract

This document describes the DecisionAnalysisPy module, a set of computational tools in Python for Decision Analysis. The various classes and functions are provided and examples from the Decision Analysis courses by K.L.Poh are used to illustrate their applications in performing various decision analysis computation tasks. This set of tools in Python supplements the other computational tools in Excel, DPL, and YAAHP which are covered in the courses. The source code for the DecisionAnalysisPy module and all the examples may be downloaded from the class website.

Contents

1 Introduction

This document describes the various Python Classes and Functions in the DecisionAnalysisPy module. Examples from the Decision Analysis course are use to illustrate how to use the code.

The souce code for the module and all the examples may be downloaded from the class website in both .py or .jpynb formats. They have been tested on a Python 3.9.x environment installed with the Andaconda distribution.

1.1 Getting Started

To use the DecisionAnalysisPy module, simply unzip the souce file DecisionAnalysisPy.py and copy it to the same directory or folder where you save your Python .py or Jupyter notebook .ipynb files. No PIP or Conda installlataion is needed.

The packages listed below that are commonly used in numerical computing, data analytics, probability & statistical computing are required. You may have to install them yourself if your Python development distribution did not include them.

- numpy
- matplotlib.pyplot
- pandas
- scipy.integrate
- scipy.optimize
- scipy.stats

In addition, the following packages are used:

- warnings
- fractions
- cycler.cycler

1.2 Classes and Functions Dependency

The follow classes and functions are available in DecisionAnalysisPy:

- risky_deal
- plot_risk_profiles (function using risky_deal)
- is1SD function (function using risky_deal)
- is2SD function (function using risky_deal)
- ExpUtilityFunction
- DistFit_continuous

- DistFit_discrete
- norm_2p
- OneWayRange Sensit
- AHPmatrix
- AHP3Lmodel (uses AHPmatrix)
- AHP4Lmodel (uses AHPmatrix)
- AHPratings_model (uses AHPmatrix)

Please report any bugs or suggestions for bug fixes to Prof. KL Poh. Suggestions for enhancements to existing code or new features are also welcome.

2 Class RiskDeal

2.1 Documentation

[1]:
```
from DecisionAnalysisPy import RiskDeal
```

[2]:
```
print(RiskDeal.__doc__)
```

```
 Class for risky deals with single-stage (flatten) probability
    distribution:

    RiskDeal(x, p, states=None, name='unnamed')
    Parameters:
      x = list or array of payoff values of deal
      p = list or array of probabilities of deal
      states = list of state names of deal
      name   = name of deal

    Attributes:
      x = np.array of payoff values
      p = np.array of probabilities
      states = list of state names; default = [x0, x1, x2, ...]
      name = name of deal, default = 'unnamed'
      EV = expected value of deal

    Methods:
      cdf(z):
          Compute CDF(z) of deal

      plot_CDF(plot_range, num=1000, dpi=100):
          Plot the CDF of the deal
          Parameters:
            plot_range = tuple(xL, xH (included), step)
                       = parameters for x-axis, e.g. (-10, 110, 5)
            num = Number of points to plot the graph (default 1000)
            dpi = dpi to plot

      plot_EPF(plot_range, num=1000, dpi=100):
          Plot the EPF of the deal
          Parameters:
            plot_range = tuple(xL, xH (included), step)
                       = parameters for x-axis, e.g. (-10, 110, 5)
            num = Number of points to plot the graph (default 1000)
            dpi = dpi to plot

       PISP(w0, uw, uw_inv=None, method='hybr', guess=None,
                 silent=False):
           Compute Personal Indifferent Selling Price of Deal
           Parameters:
```

```
            w0 = intial wealth
            uw = wealth utility function
            uw_inv = inverse wealth utility function;
                      If not provided, an equation solver will be used.
            method = solver method used: 'hybr' (default) or 'lm'
            guess = starting point for solver
            silent = False (default): print message; True: no message

      CE(w0, uw, uw_inv=None, method='hybr', guess=None,
               silent=False):
          Compute Certainty Equivalent of deal = PISP

      PIBP(w0, uw, method='hybr', guess=None, silent=False):
          Compute personal indifferent buying price of Deal.
          Parameters:
              w0 = intial wealth
              wu = wealth utility function
              method = solver method used: 'hybr' (default), 'lm'
              guess = starting point for solver
              silent = False (default): print message; True: no message
```

[ ]:

```

```

2.2 Risk Profiles Plotting Function

2.2.1 Plot Party Problem Risk Profiles

Source: 4.3_PartyProblem_Plot_Risk_Profiles_with_RiskDeal_Class.ipynb

[1]:

```
""" Plotting Risk Profiles for the Party Problem """
from DecisionAnalysisPy import RiskDeal
from DecisionAnalysisPy import plot_risk_profiles
```

[2]:

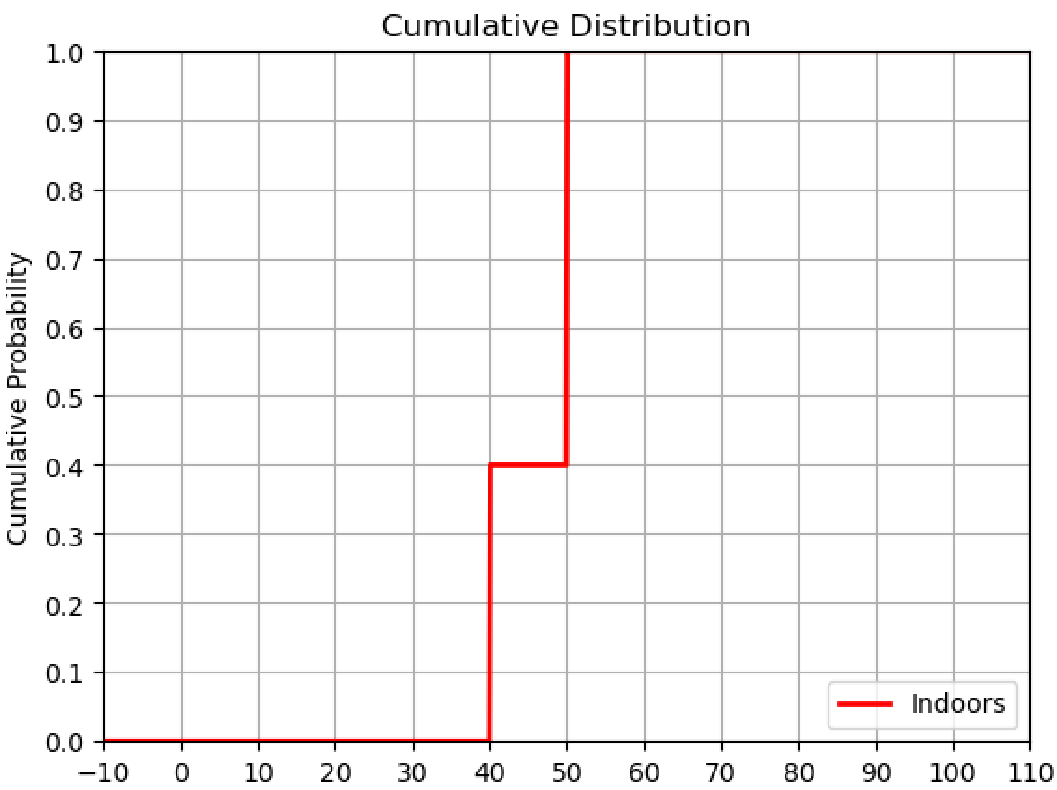
```
# Create the 3 risky deals
ID = RiskDeal(x=[40,   50],
              p=[0.4, 0.6],
              states=['sunny','rainy'],
              name='Indoors')

PR = RiskDeal(x=[90,   20],
              p=[0.4, 0.6],
              states=['sunny','rainy'],
              name='Porch')

OD = RiskDeal(x=[100,   0],
              p=[0.4, 0.6],
```

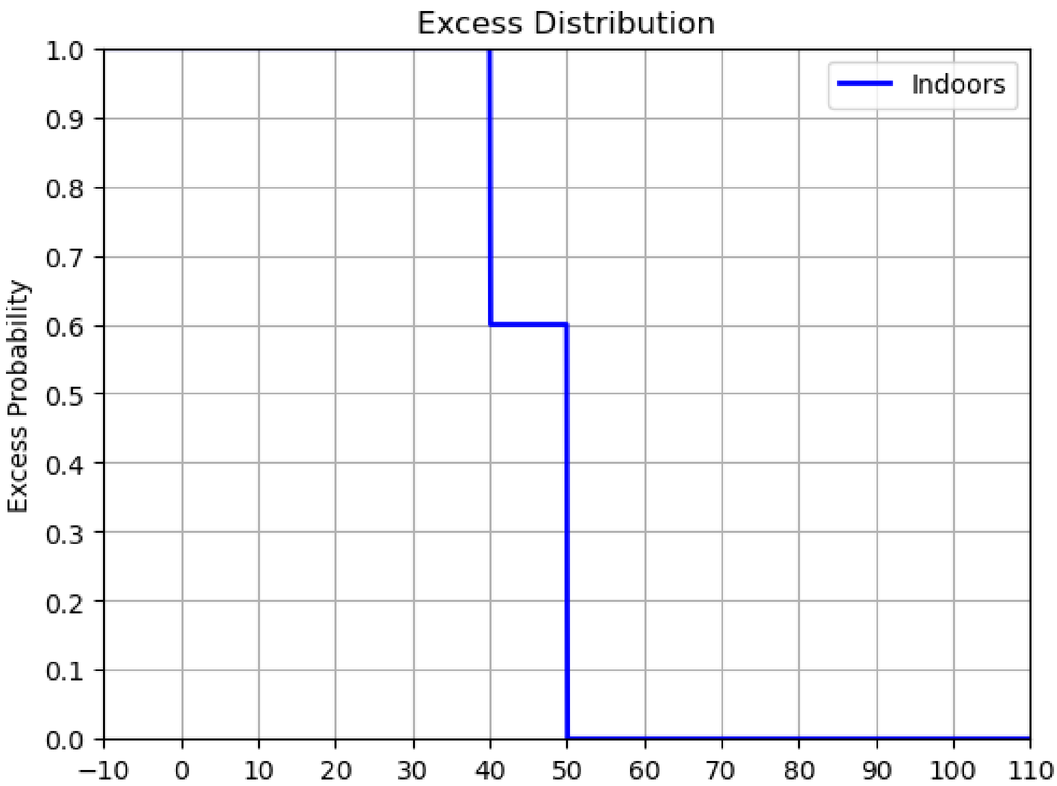
```
                 states=['sunny','rainy'],
                 name='Outdoors')
```

[3]:
```
# Parameters for plotting risk profiles
plot_range=(-10, 110, 10)
npoints = 1000
```

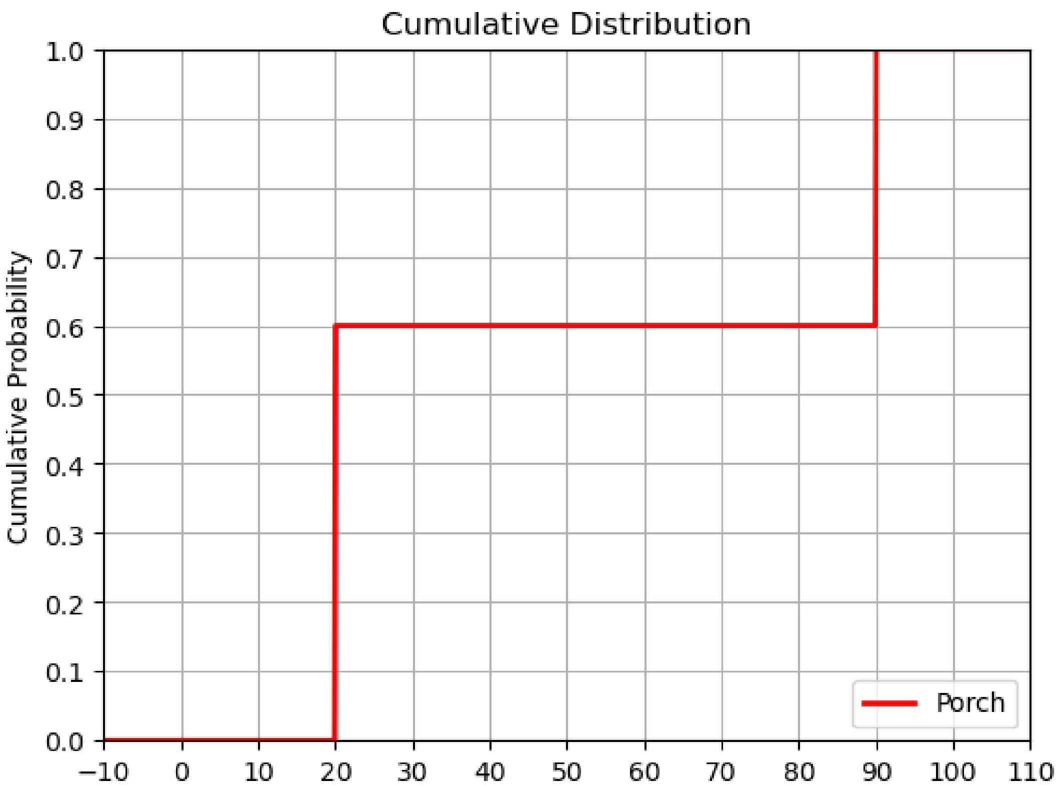
[4]:
```
# Plot the 3 risk profiles individually
for deal in [ID, PR, OD]:
    print(f"\nRisk Profiles for {deal.name}:")
    deal.plot_CDF(plot_range, npoints, dpi=100)
    deal.plot_EPF(plot_range, npoints, dpi=100)
```

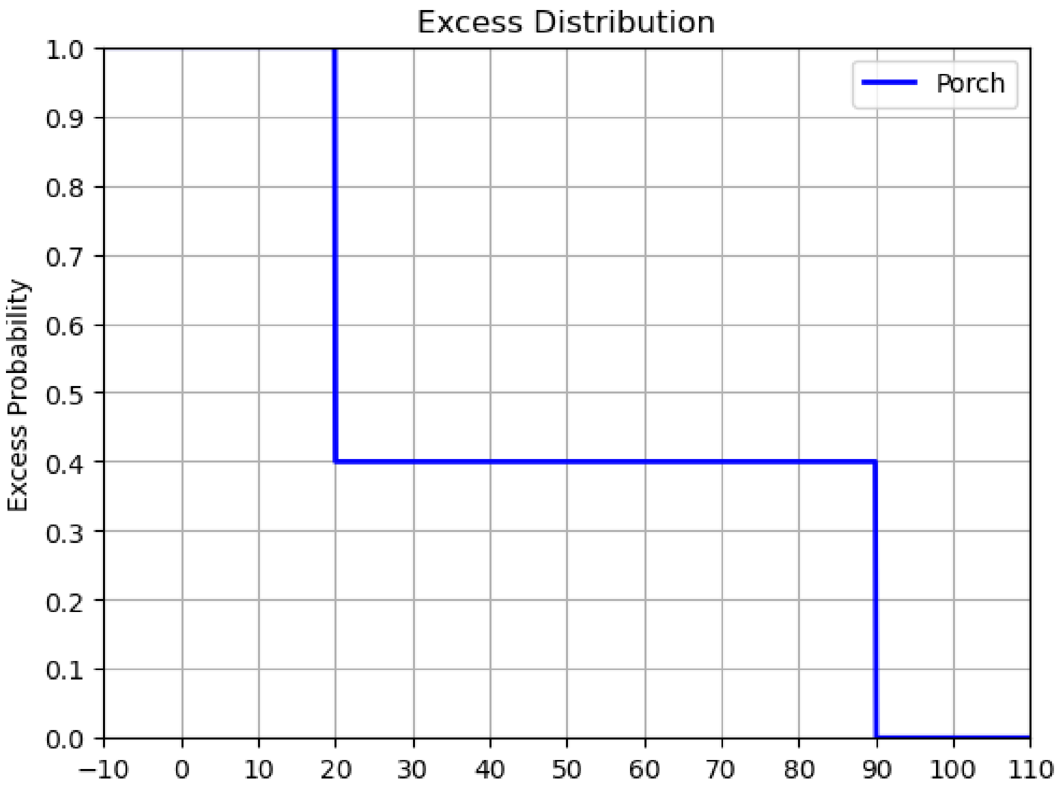
```
Risk Profiles for Indoors:
```



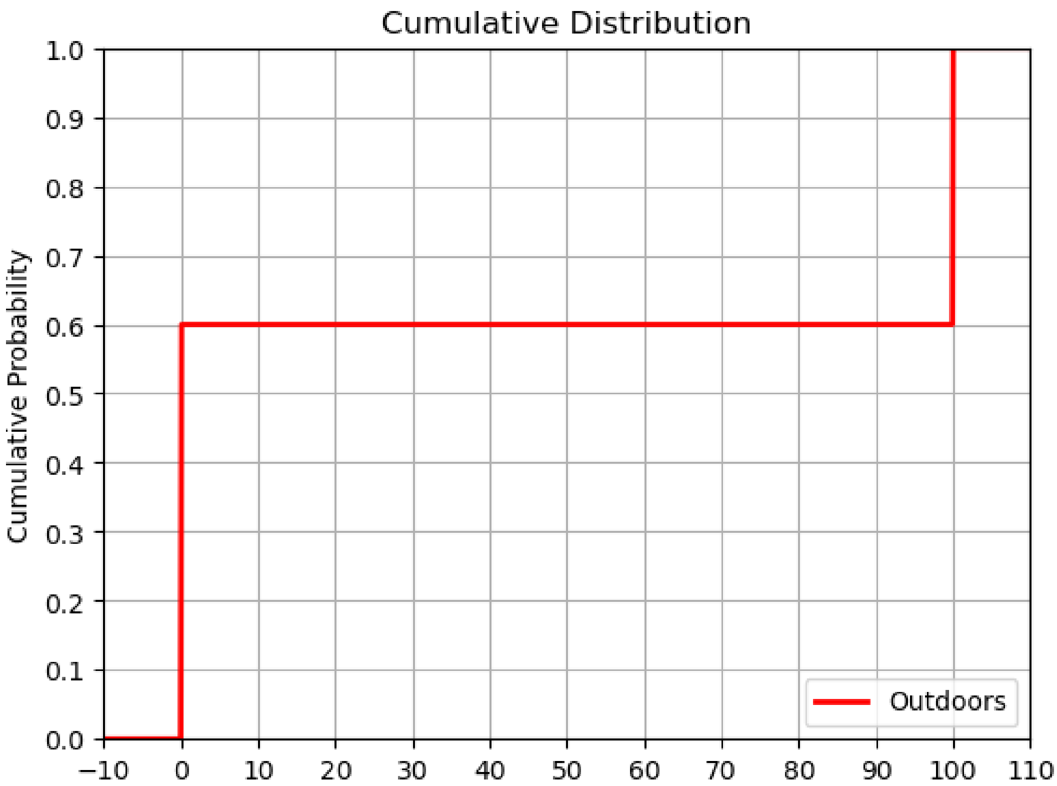


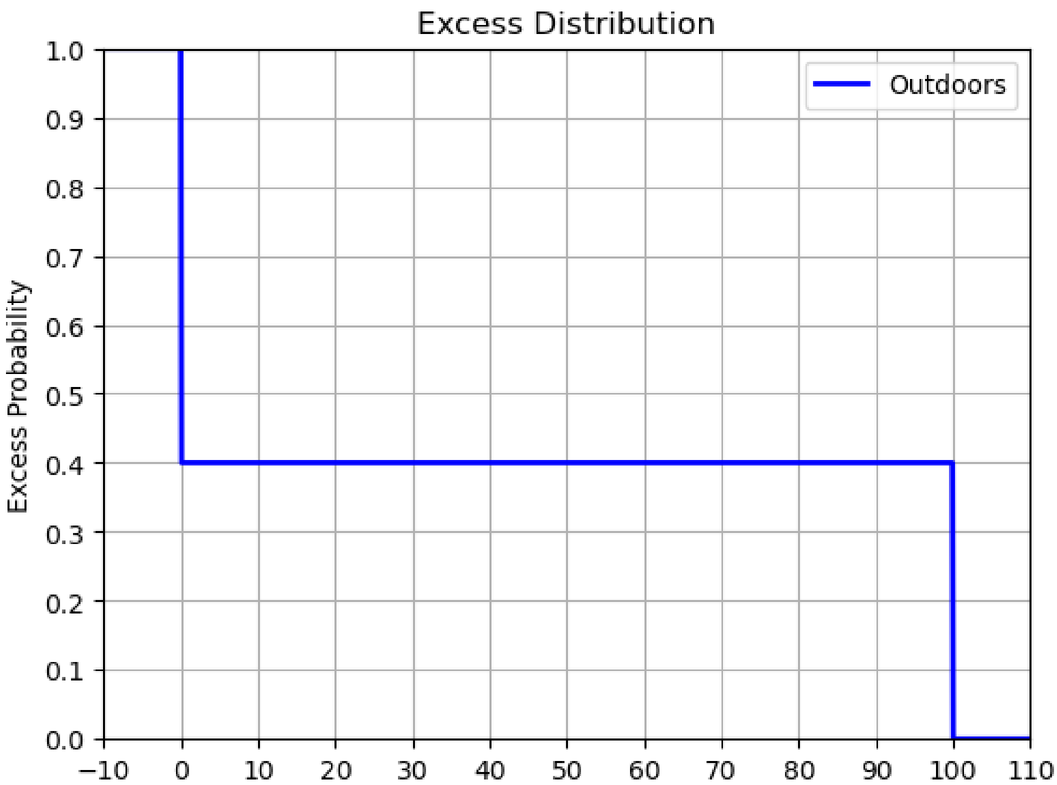
Risk Profiles for Porch:





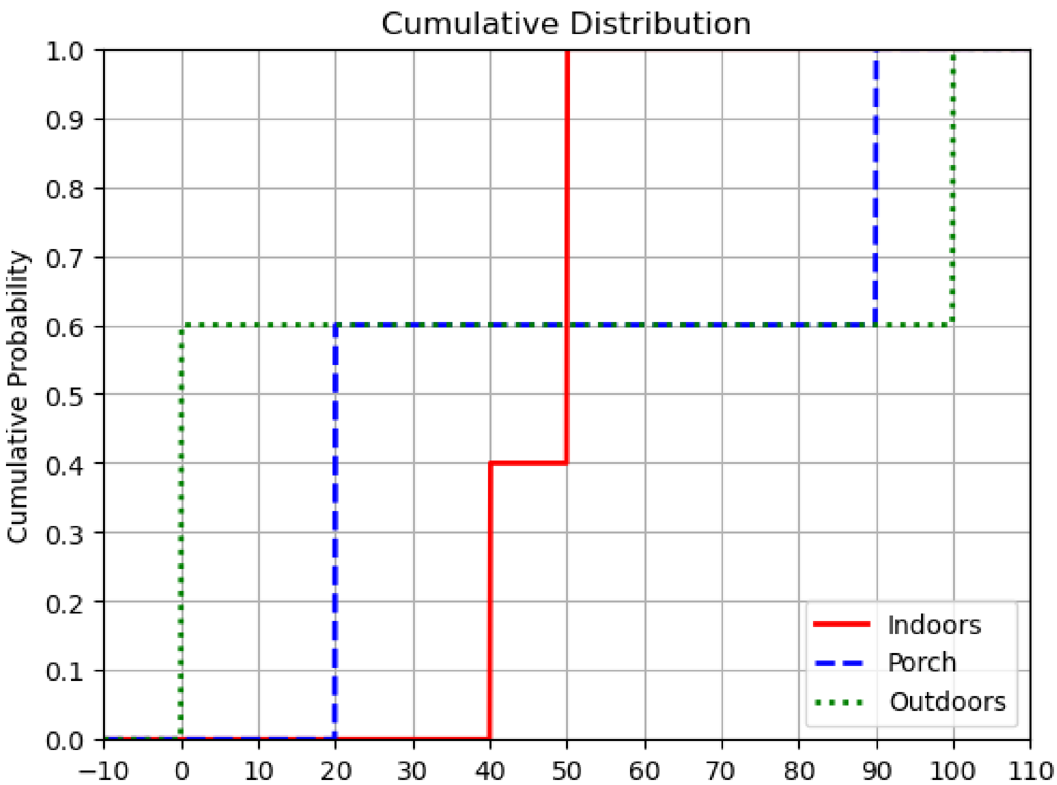
Risk Profiles for Outdoors:

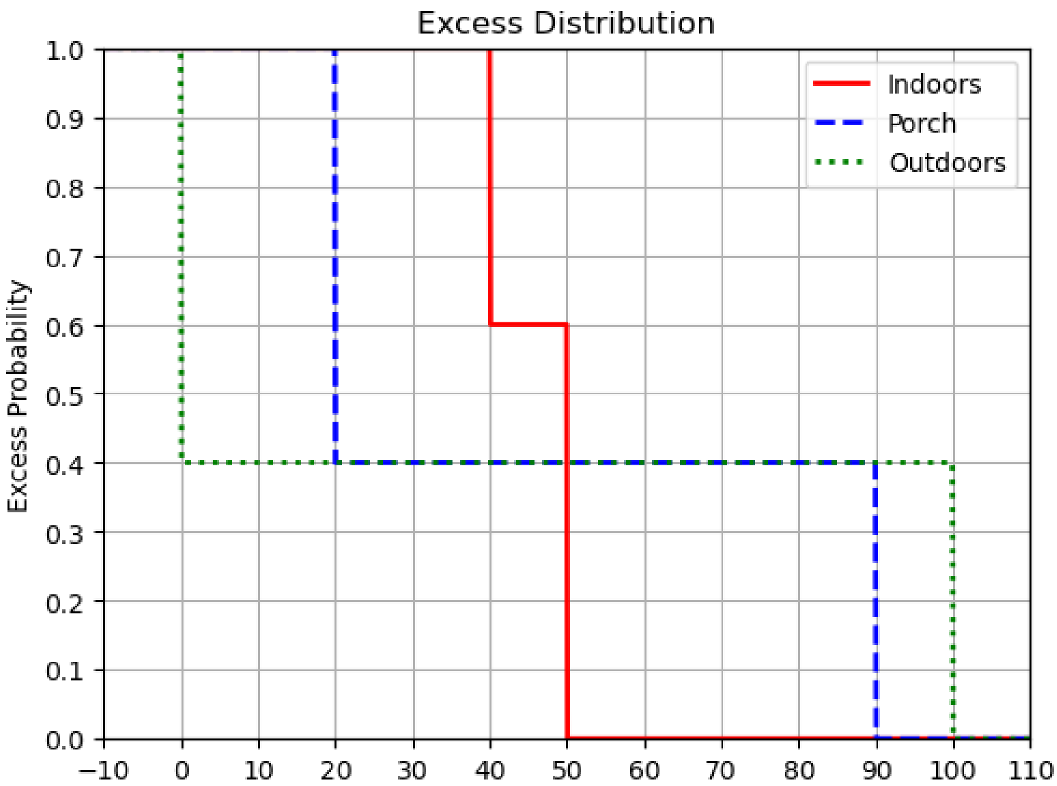

Cumulative Distribution
Cumulative Probability
Outdoors
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10
0
10
20
30
40
50
60
70
80
90
100
110


Excess Distribution
Excess Probability
Outdoors
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10
0
10
20
30
40
50
60
70
80
90
100
110

```
[5]: # Plot all the risk profiles together for comparison
print("\nRisk Profiles for the Party Problem")
plot_risk_profiles([ID, PR, OD], plot_range,
                   num=npoints, CDF=True, EPF=True, dpi=100)
```

```
Risk Profiles for the Party Problem
```





[ ]:

2.3 Stochastic Dominance Analysis Functions

2.3.1 Party Problem: First and Second Order Stochastic Dominance Analysis

Source: 4.7_PartyProblem_Stochastic_Dominance_using_RiskDeal_Class.ipynb

[1]:
```
""" Stochastic Dominance Analysis for the Party Problem using
    RiskDeal Class and is1SD & is2SD functions """
from DecisionAnalysisPy import RiskDeal
from DecisionAnalysisPy import plot_risk_profiles, is1SD, is2SD
```

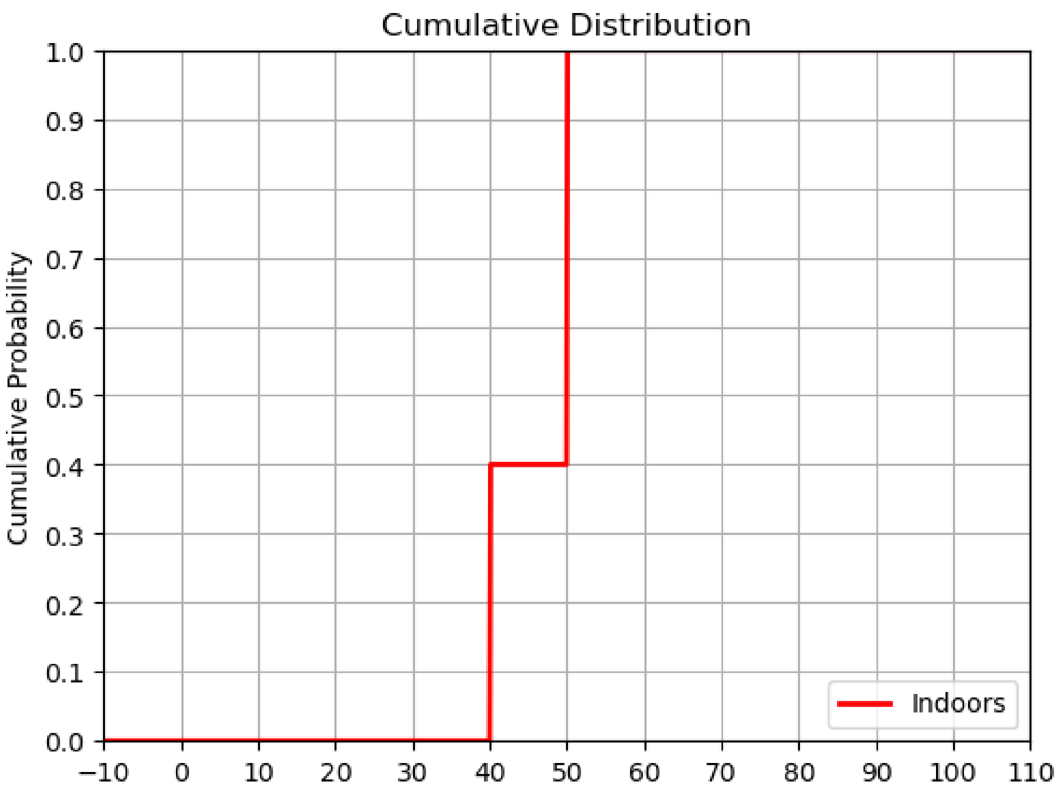
[2]:
```
# Create the 3 risky deals
ID = RiskDeal(x=[40,  50],
              p=[0.4, 0.6],
              states=['sunny','rainy'],
              name='Indoors')

PR = RiskDeal(x=[90,  20],
              p=[0.4, 0.6],
              states=['sunny','rainy'],
              name='Porch')
```

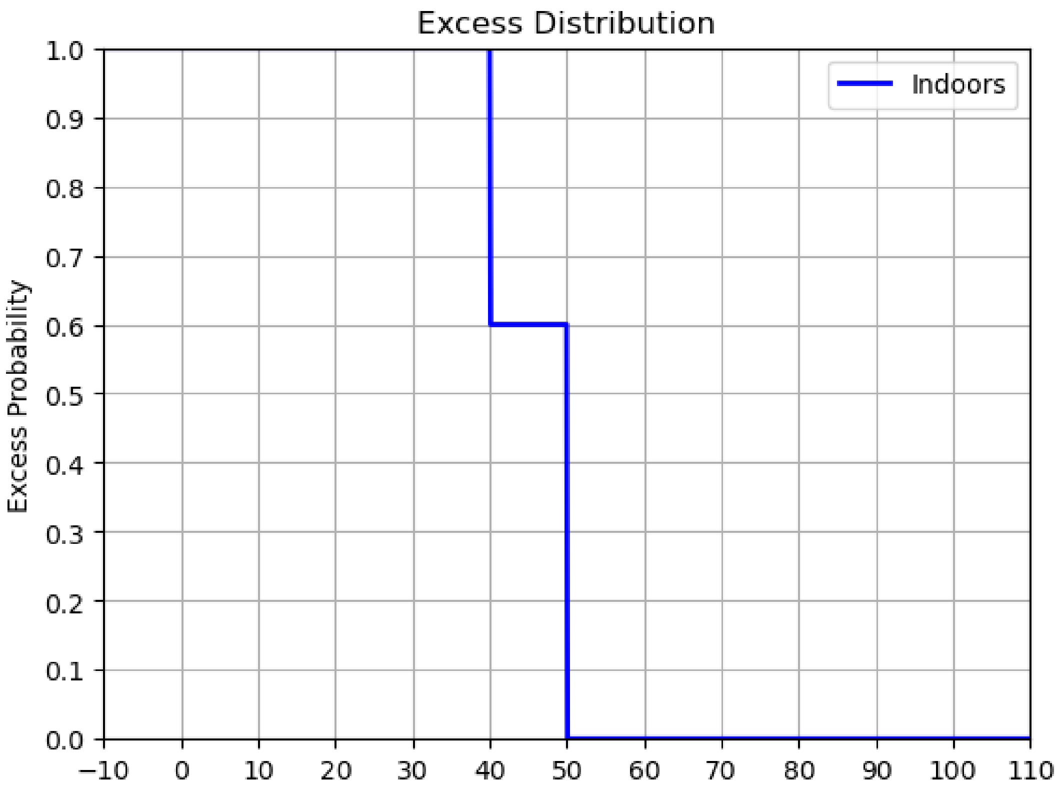
```
OD = RiskDeal(x=[100, 0],
              p=[0.4, 0.6],
              states=['sunny','rainy'],
              name='Outdoors')
```

[3]:
```
# Parameters for plotting and checking stochastic dominance
plot_range=(-10, 110, 10)
npoints = 1000
```

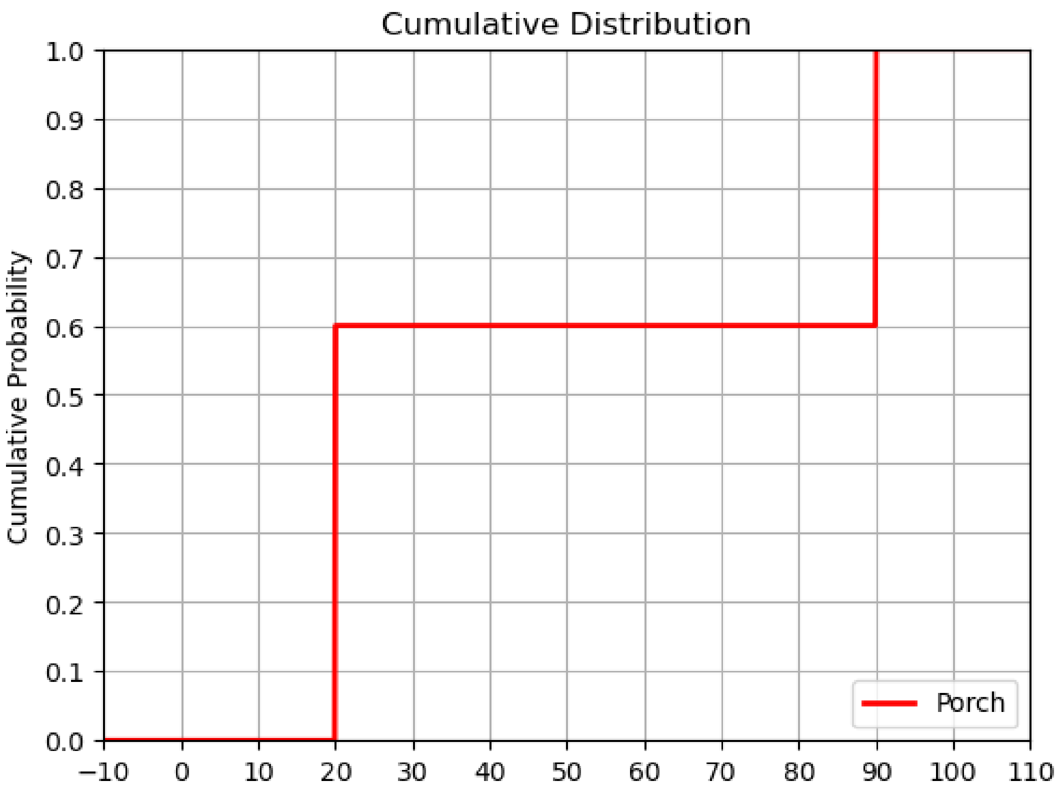
[4]:
```
# Plot the 3 risk profiles individually
for deal in [ID, PR, OD]:
    print(f"\nRisk Profiles for {deal.name}:")
    deal.plot_CDF(plot_range, npoints, dpi=100)
    deal.plot_EPF(plot_range, npoints, dpi=100)
```

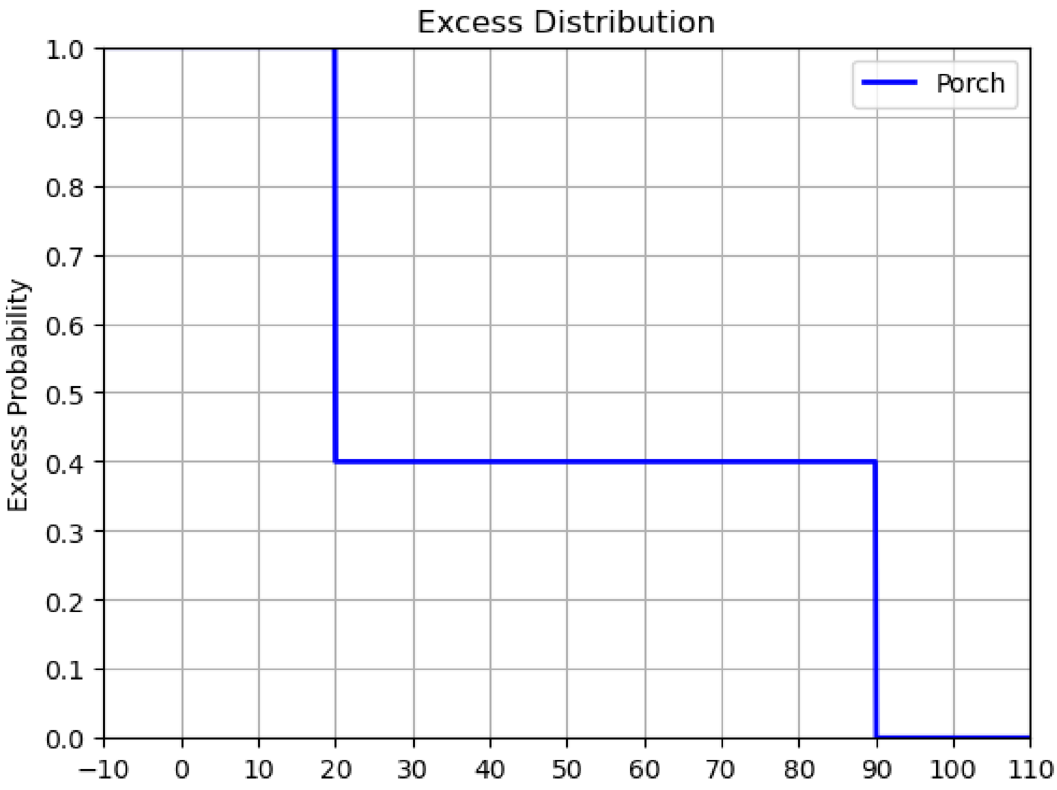
```
Risk Profiles for Indoors:
```



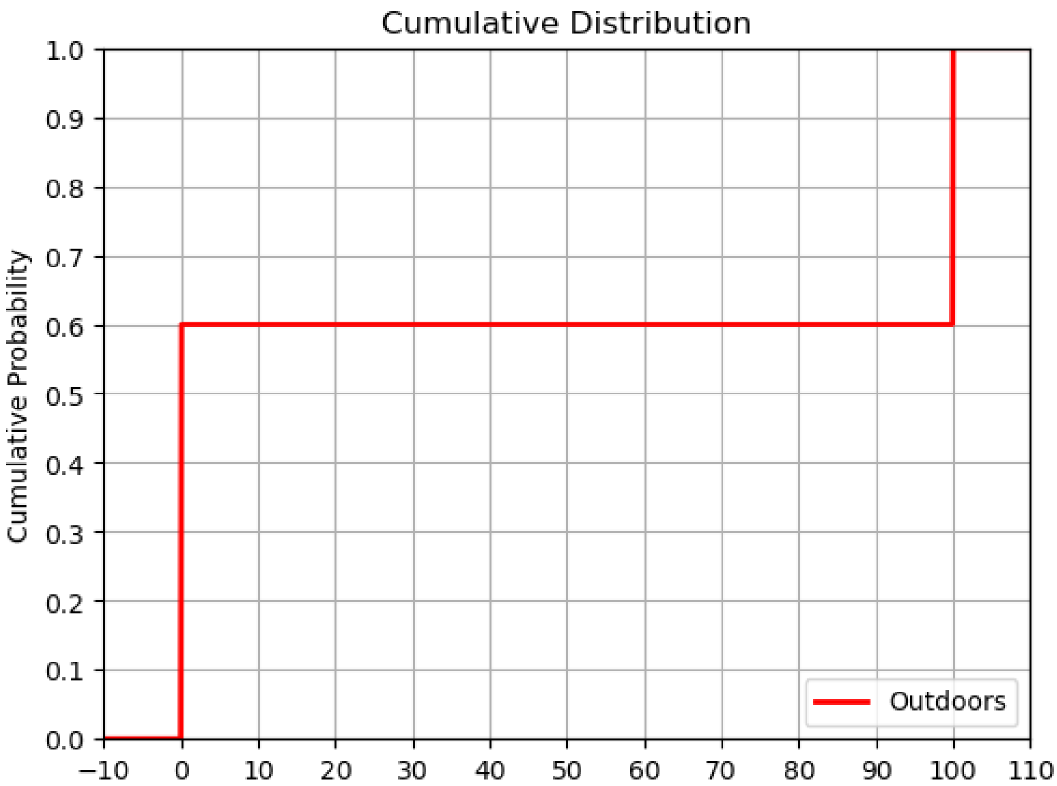


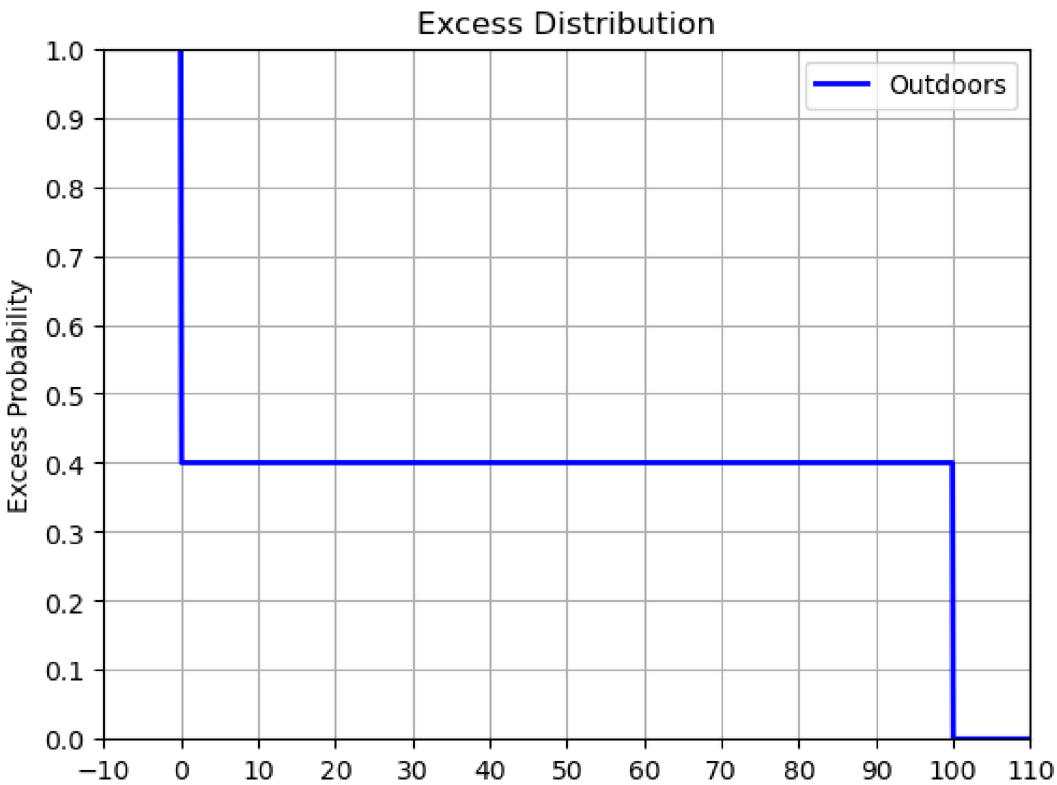
Risk Profiles for Porch:





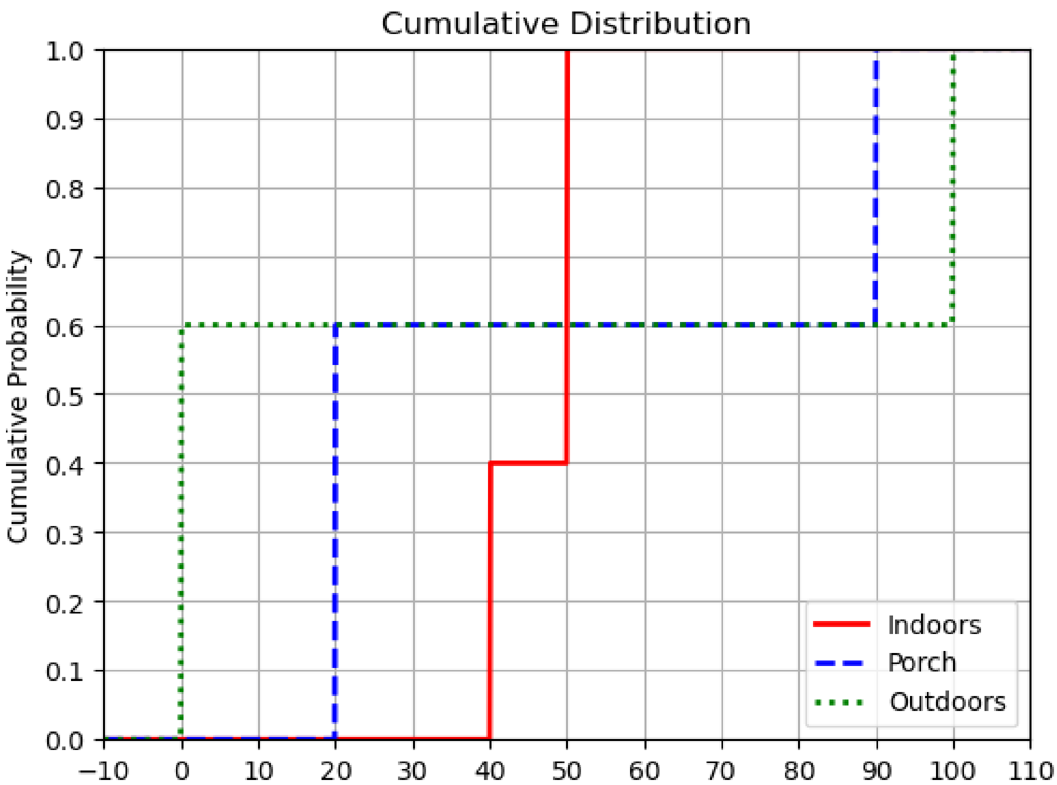

```
Risk Profiles for Outdoors:
```

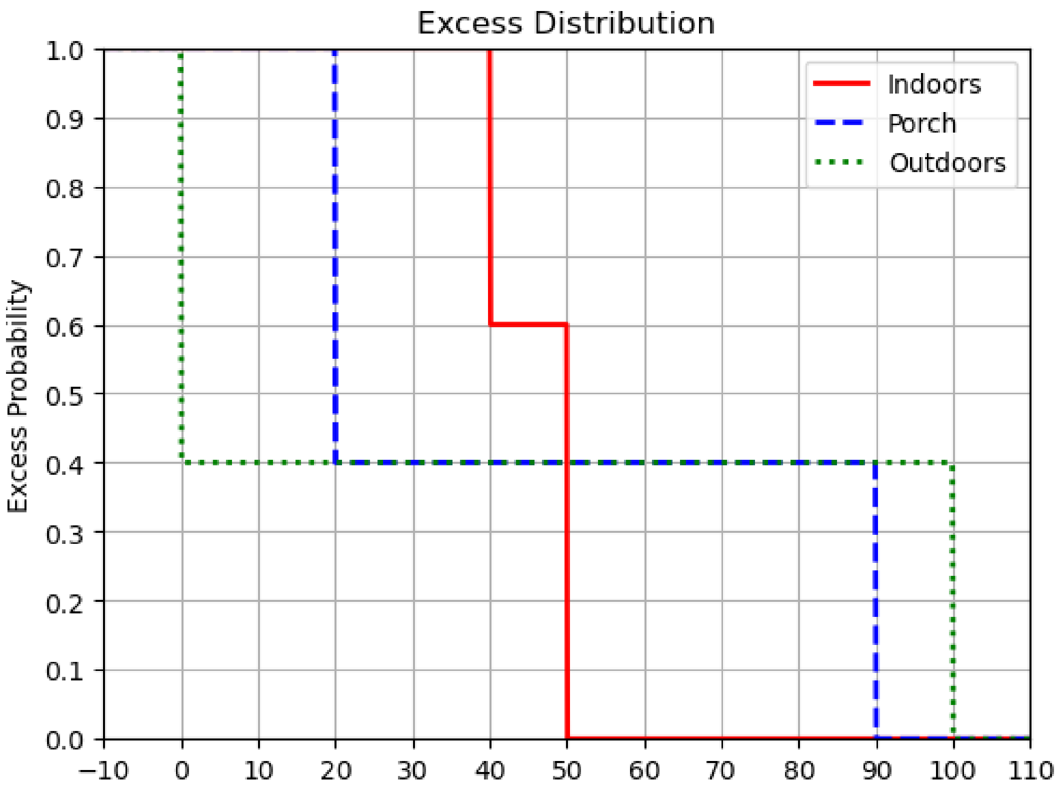

Cumulative Distribution
Cumulative Probability
Outdoors
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10
0
10
20
30
40
50
60
70
80
90
100
110


Excess Distribution
Excess Probability
Outdoors
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10
0
10
20
30
40
50
60
70
80
90
100
110

[5]:
```python
# Plot all the 3 risk profiles together for comparison
print("\nRisk Profiles for the Party Problem")
plot_risk_profiles([ID, PR, OD], plot_range,
                   num=npoints, CDF=True, EPF=True, dpi=100)
```

```
Risk Profiles for the Party Problem
```



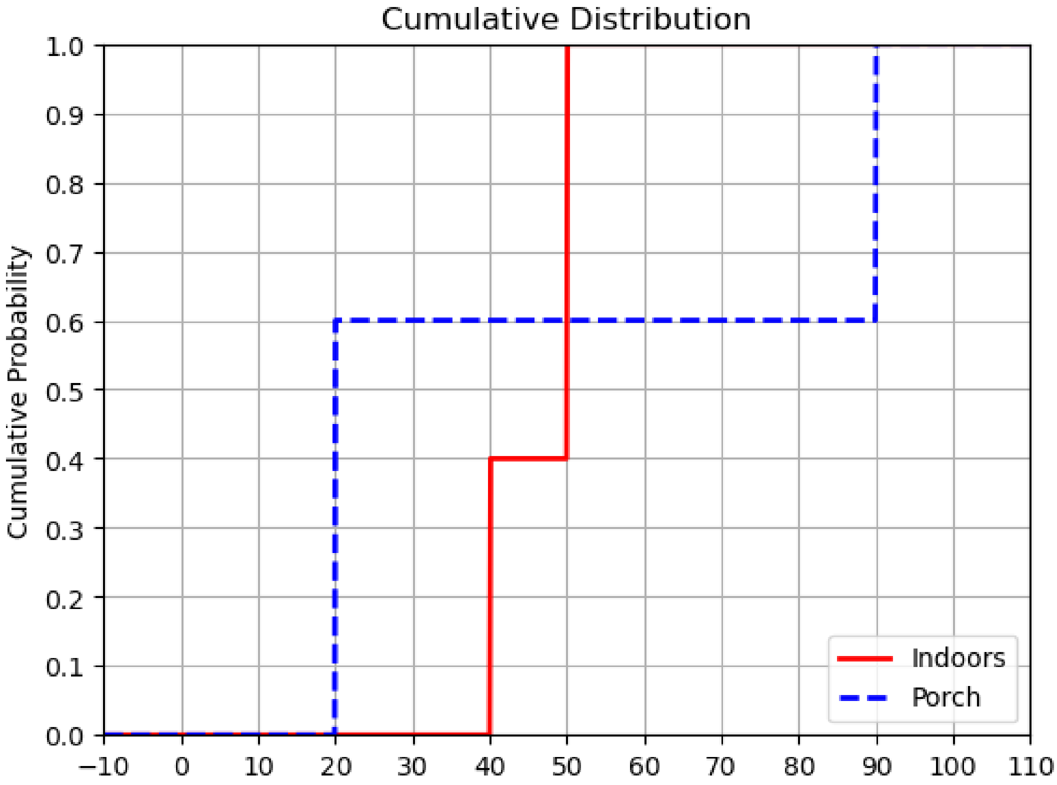


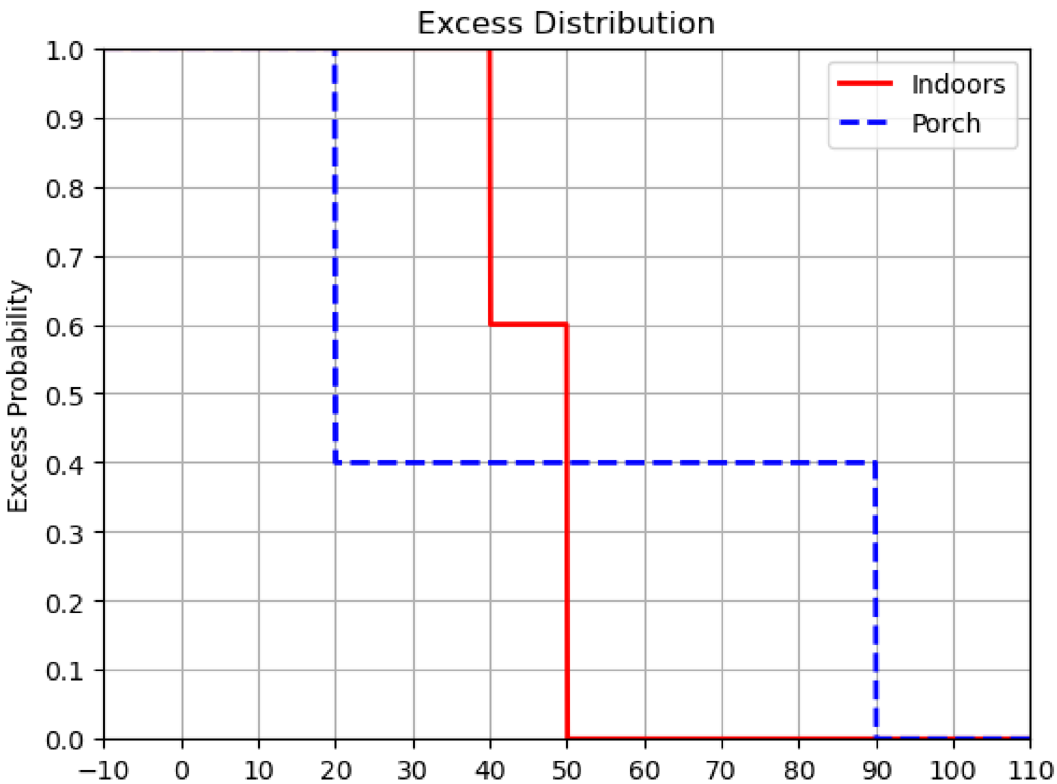
[6]:
```
# Check for 1SD using is1SD function
compare_range = plot_range[:-1]
npoints = 1000
print("\nChecking for 1st Order Stochastic Dominances:")
for A, B in [(ID, PR), (ID, OD), (PR, OD)]:
    if is1SD(A, B, compare_range, npoints):
        print(f"  {A.name} 1SD {B.name}")
    else:
        print(f"  {A.name} Does Not 1SD {B.name}")
```

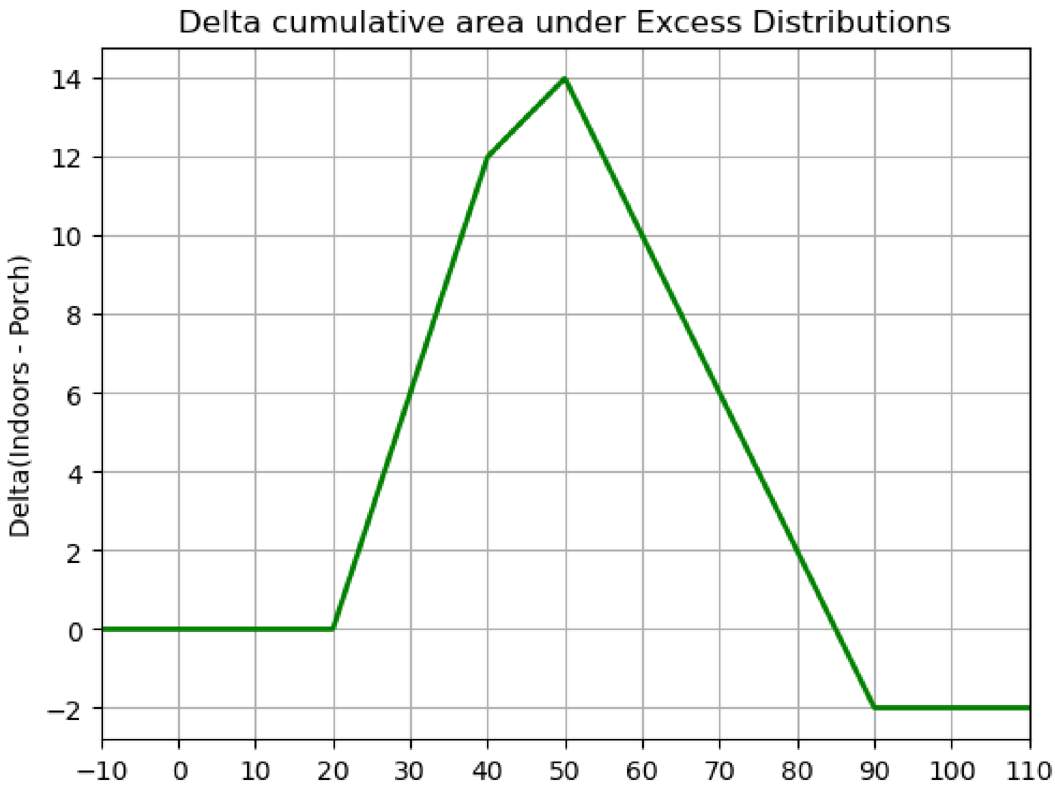
```
Checking for 1st Order Stochastic Dominances:
  Indoors Does Not 1SD Porch
  Indoors Does Not 1SD Outdoors
  Porch Does Not 1SD Outdoors
```

[7]:
```
# Check for 2SD using is2SD function
for A, B in [(ID, PR), (ID, OD), (PR, OD)]:
    print(f"\nChecking if {A.name} 2SD {B.name}:")
    if is2SD(A, B, plot_range, npoints, show_plot=True, dpi=100):
        print(f"\n{A.name} 2SD {B.name}")
    else:
        print(f"\n{A.name} Does Not 2SD {B.name}")
```

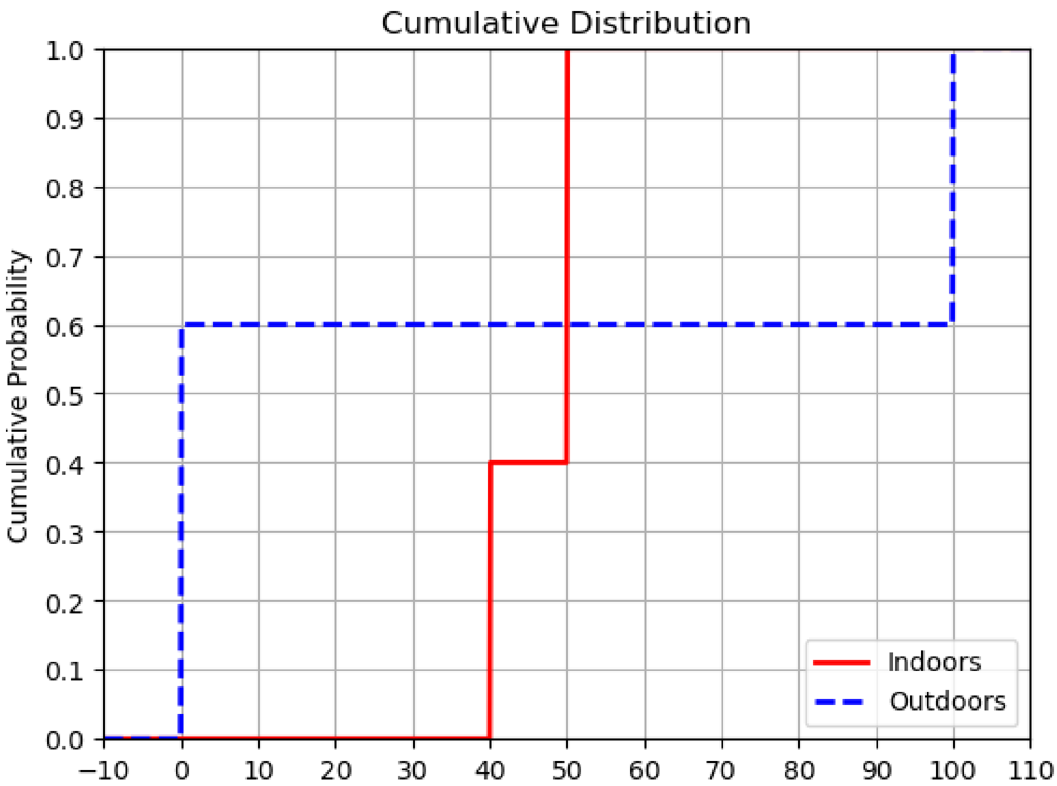
```
Checking if Indoors 2SD Porch:
```

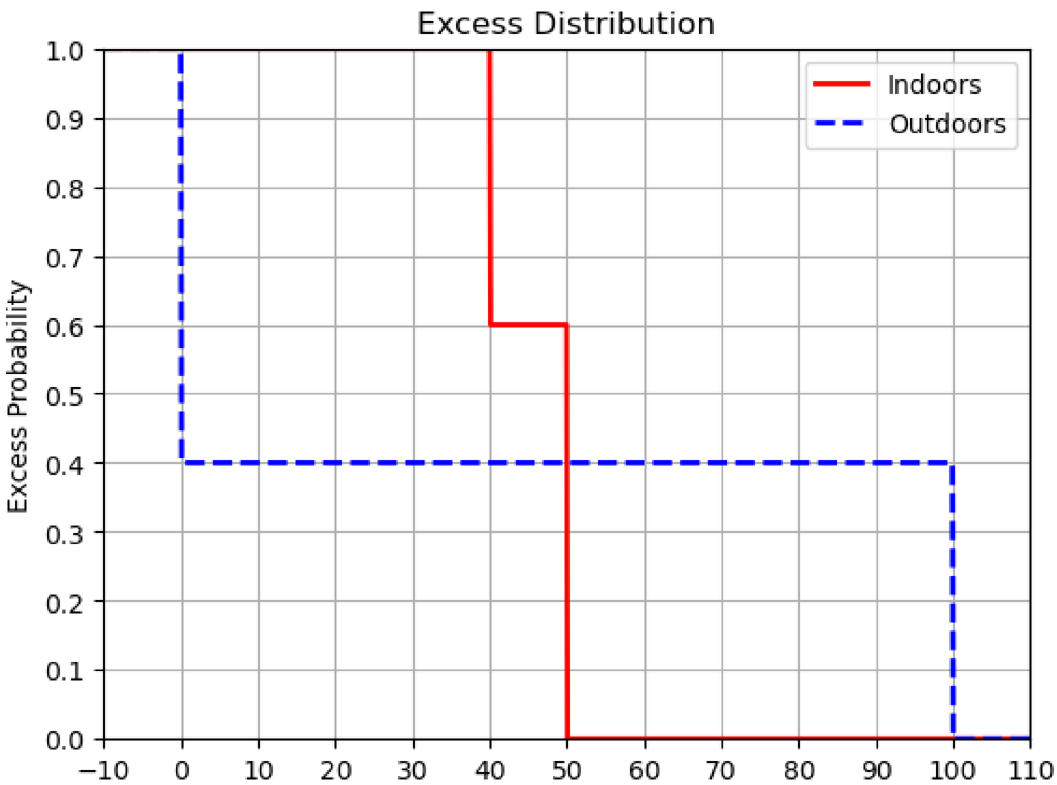


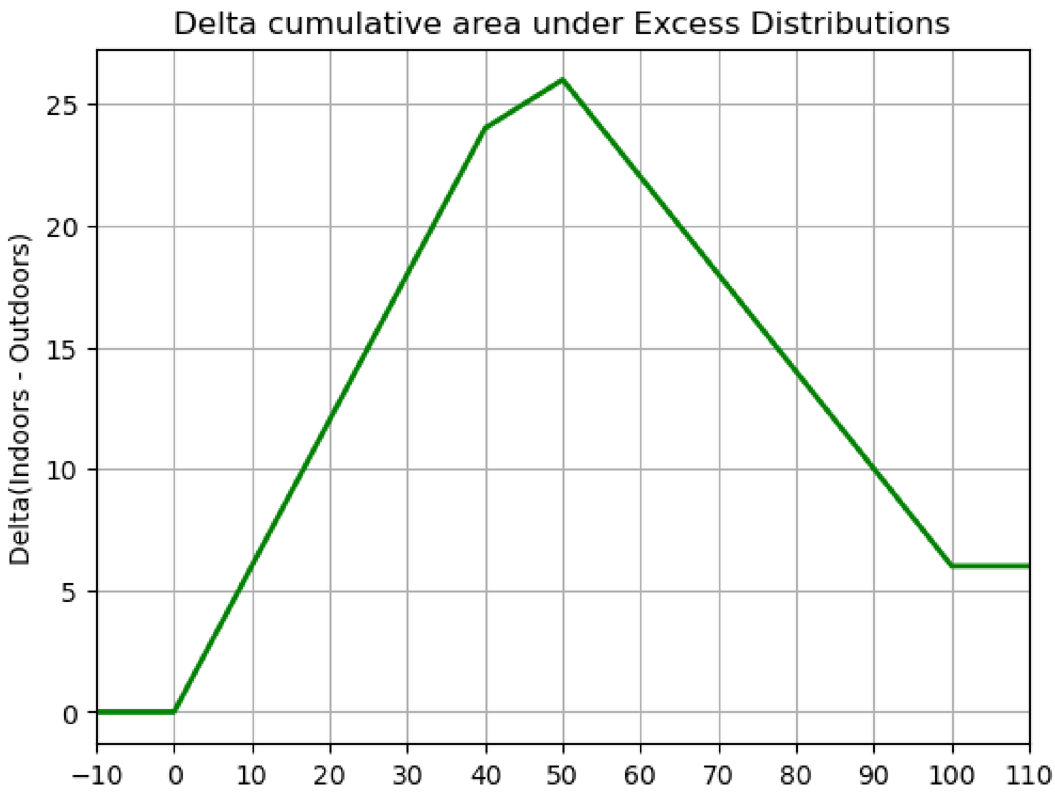




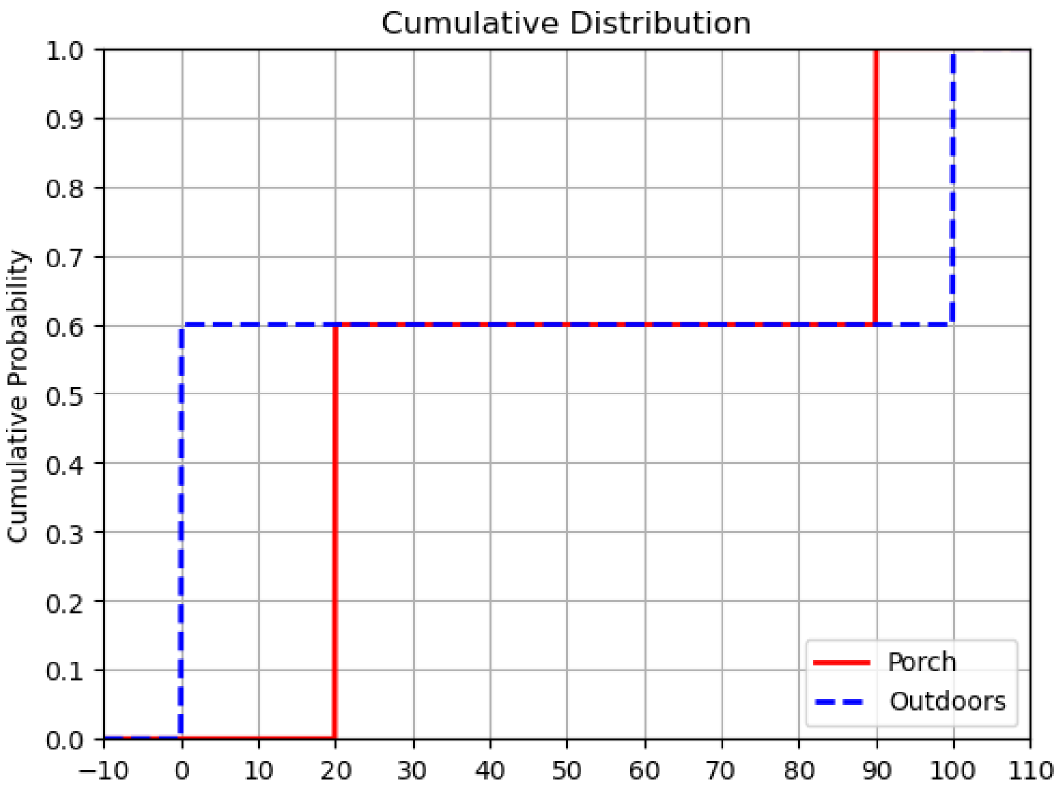

```
Indoors Does Not 2SD Porch

Checking if Indoors 2SD Outdoors:
```

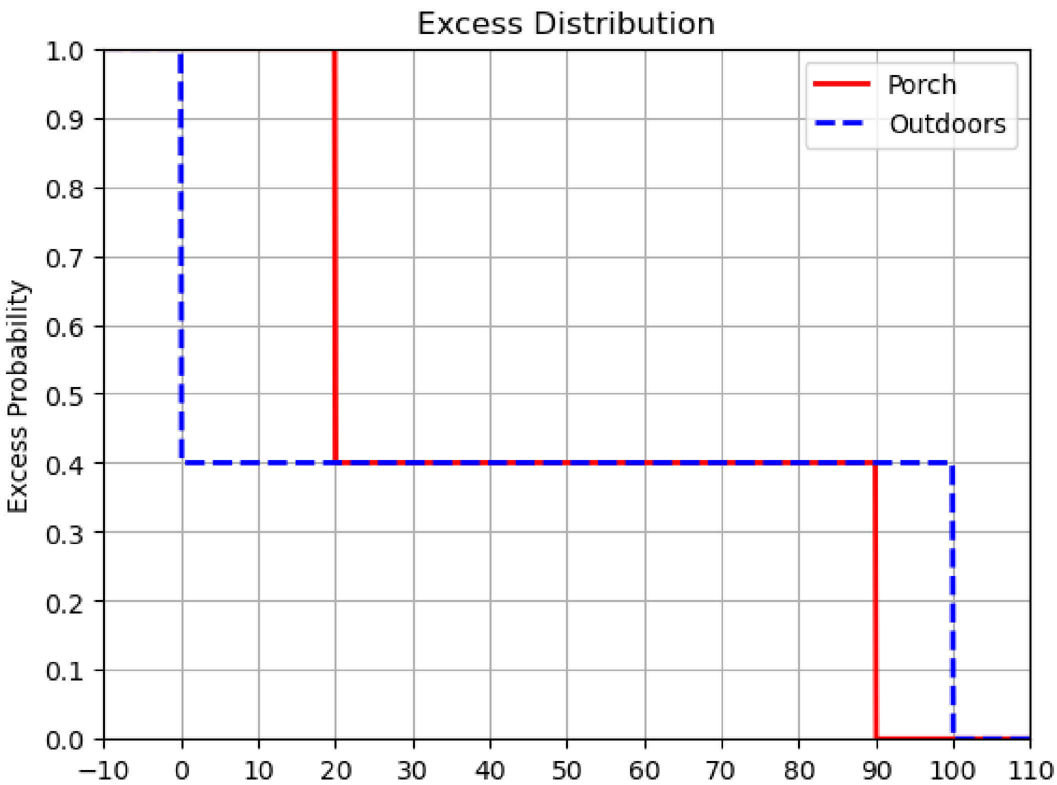

Cumulative Distribution
Cumulative Probability
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10
0
10
20
30
40
50
60
70
80
90
100
110
Indoors
Outdoors

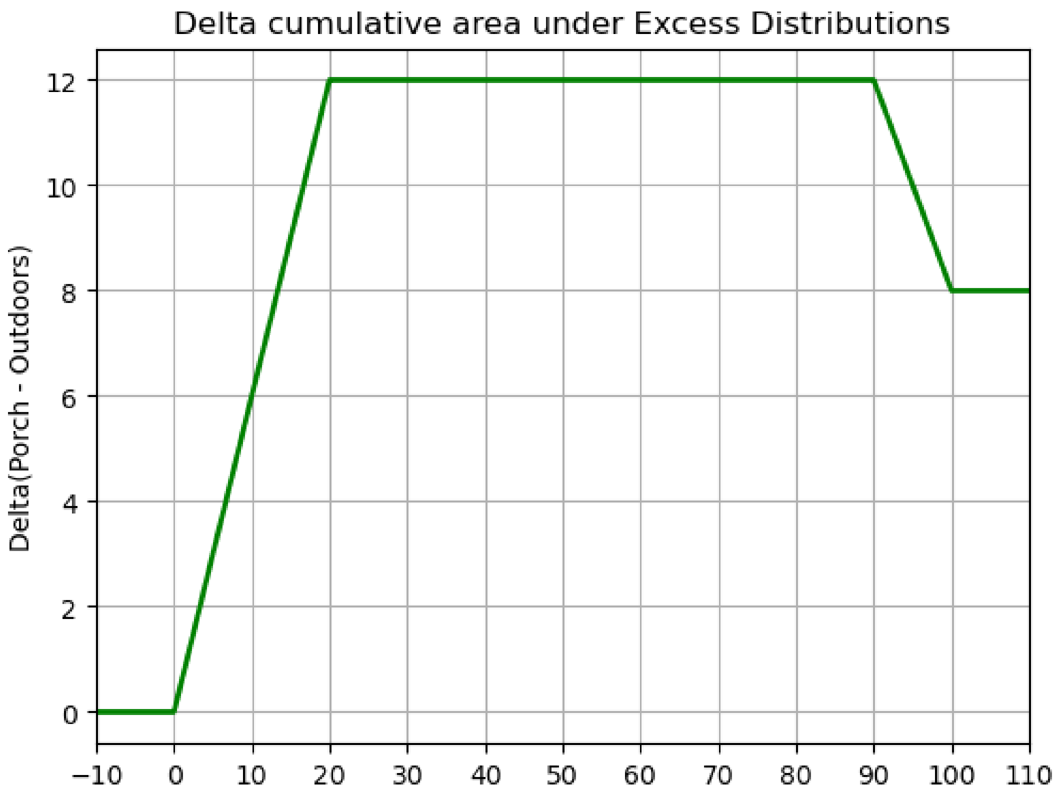

Excess Distribution
Excess Probability
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10
0
10
20
30
40
50
60
70
80
90
100
110
Indoors
Outdoors




```
Indoors 2SD Outdoors

Checking if Porch 2SD Outdoors:
```


Cumulative Distribution
Cumulative Probability
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10
0
10
20
30
40
50
60
70
80
90
100
110
Porch
Outdoors


Excess Distribution
Excess Probability
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10
0
10
20
30
40
50
60
70
80
90
100
110
Porch
Outdoors




```
Porch 2SD Outdoors
```

[ ]:

```

```

2.3.2 Exxoff Problem: Stochastic Dominance Analysis

Source: 8.2.2_Exxoff_Stochastic_Dominance_Analysis.ipynb

[1]:

```
""" Exxoff Problem: Stochastic Dominance Analysis """
from DecisionAnalysisPy import RiskDeal
from DecisionAnalysisPy import plot_risk_profiles, is1SD, is2SD
```

[2]:

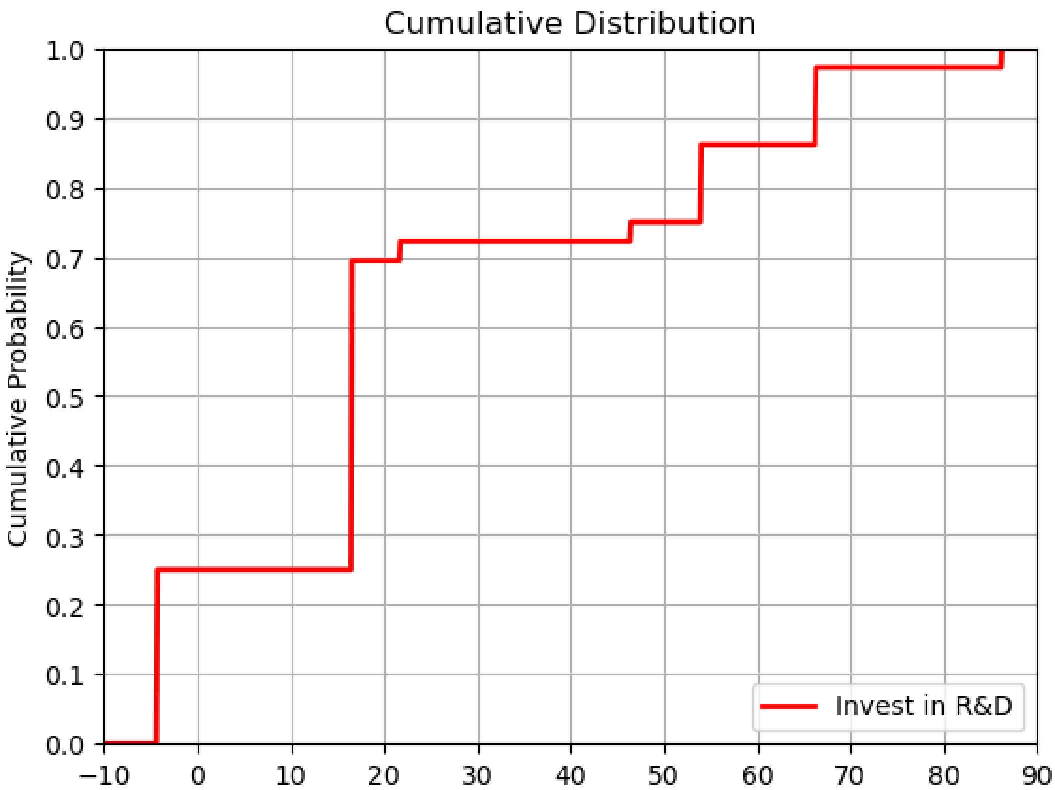
```
# create the risky deals
Invest = RiskDeal (
    x=[-4.3464, 16.4602, 21.7300, 46.3809, 53.9320, 66.2542, 86.1339 ],
    p=[0.25000,  0.4444,  0.0278,  0.0278,  0.1111,  0.1111,  0.0278 ],
    name='Invest in R&D')

No_invest = RiskDeal(x=[0], p=[ 1 ], name='Do not invest')
```

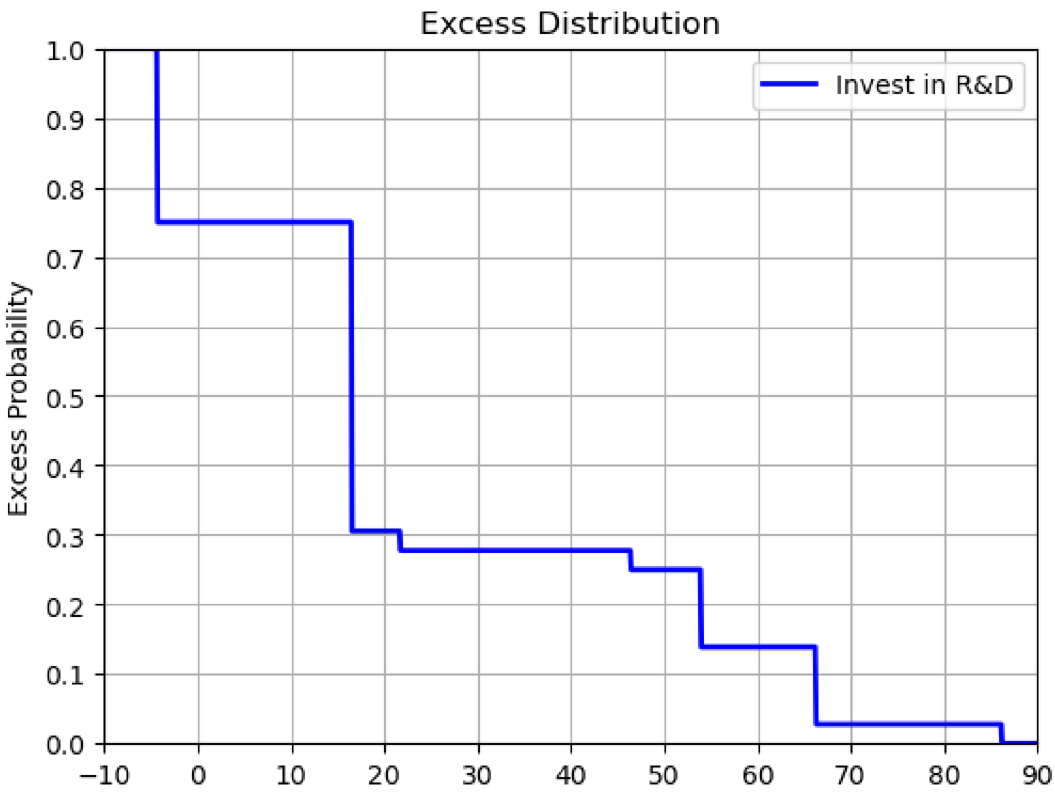
[3]:

```
# Parameters for risk profiles plotting and SD analsyis
plot_range = (-10, 90, 10)
npoints = 1000
```

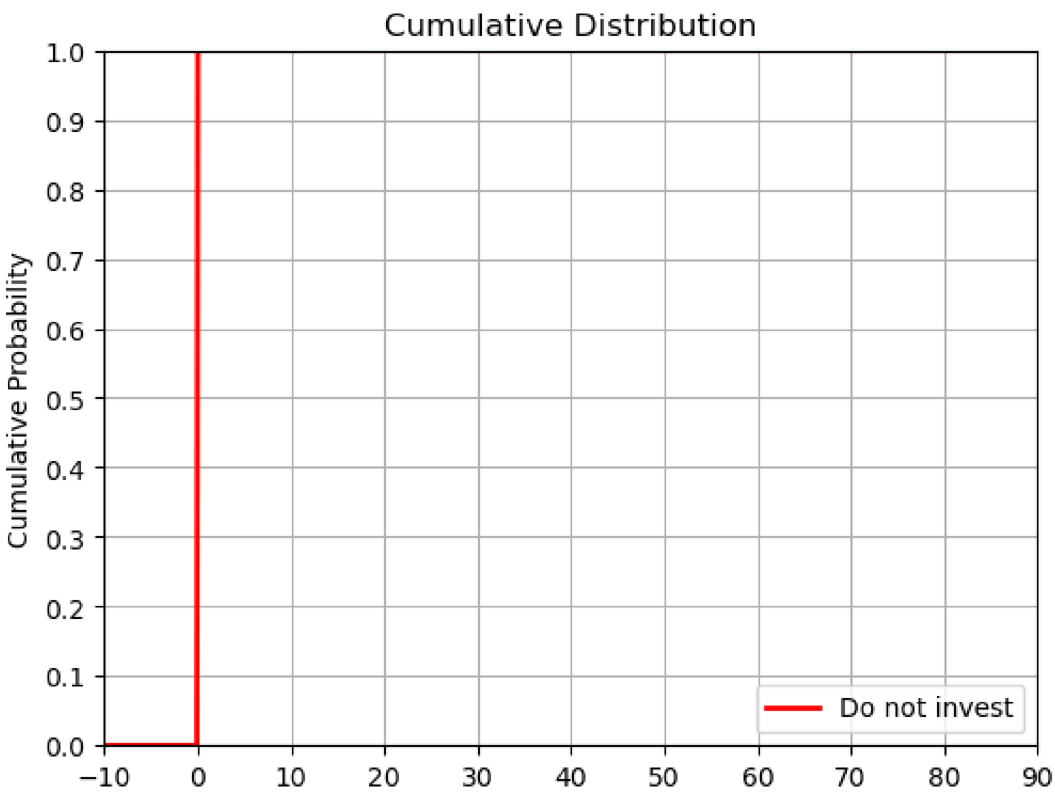
[4]:
```
# Plot the invidvual risk profiles
for deal in [Invest, No_invest]:
    print(f"\nRisk Profiles for {deal.name}:")
    deal.plot_CDF(plot_range, npoints, dpi=100)
    deal.plot_EPF(plot_range, npoints, dpi=100)
```

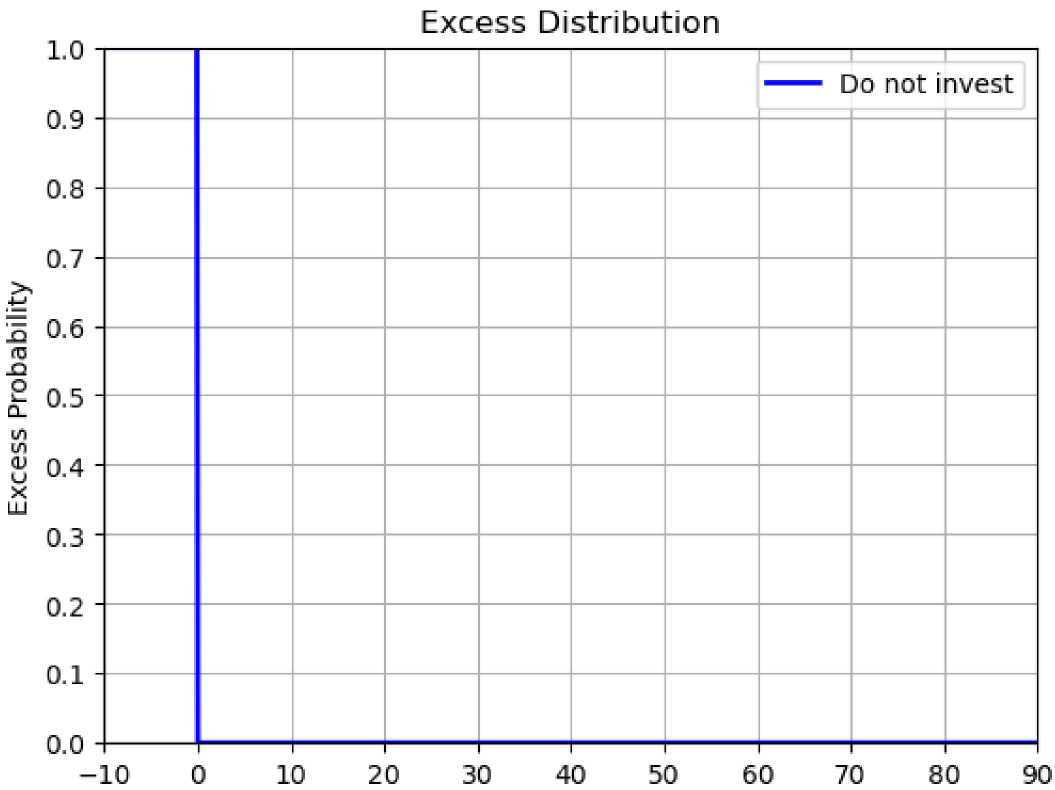
```
Risk Profiles for Invest in R&D:
```





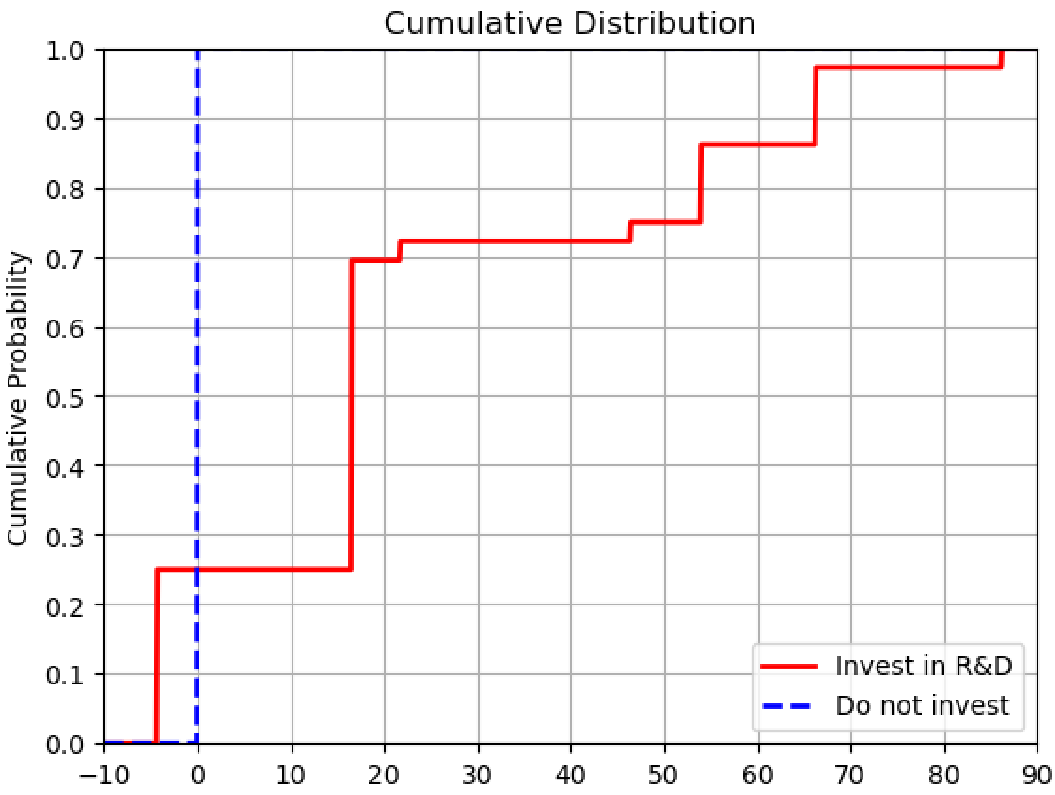

```
Risk Profiles for Do not invest:
```



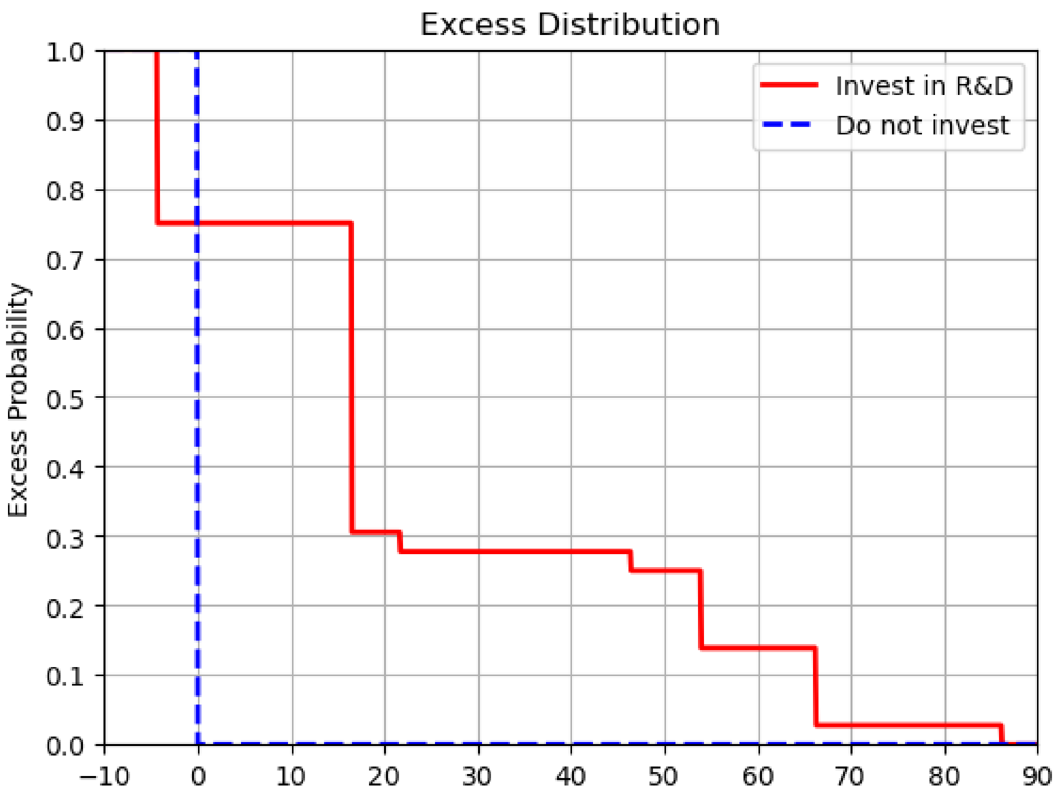


[5]:
```
# Plot the 2 risk profiles together for comparison
print("\nRisk Profiles for Exxoff Problem")
plot_risk_profiles([Invest, No_invest], plot_range, num=npoints,
                   CDF=True, EPF=True, dpi=100)
```

```
Risk Profiles for Exxoff Problem
```



Cumulative Distribution
Cumulative Probability
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10
0
10
20
30
40
50
60
70
80
90
Invest in R&D
Do not invest



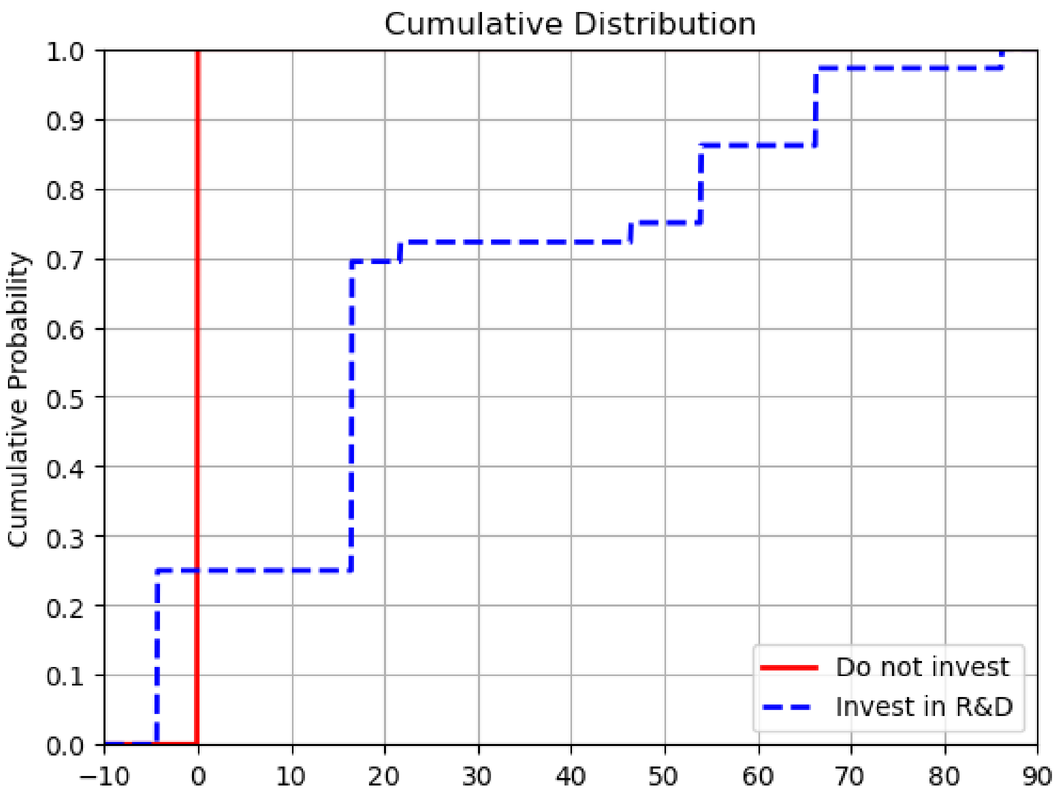
Excess Distribution
Excess Probability
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10
0
10
20
30
40
50
60
70
80
90
Invest in R&D
Do not invest

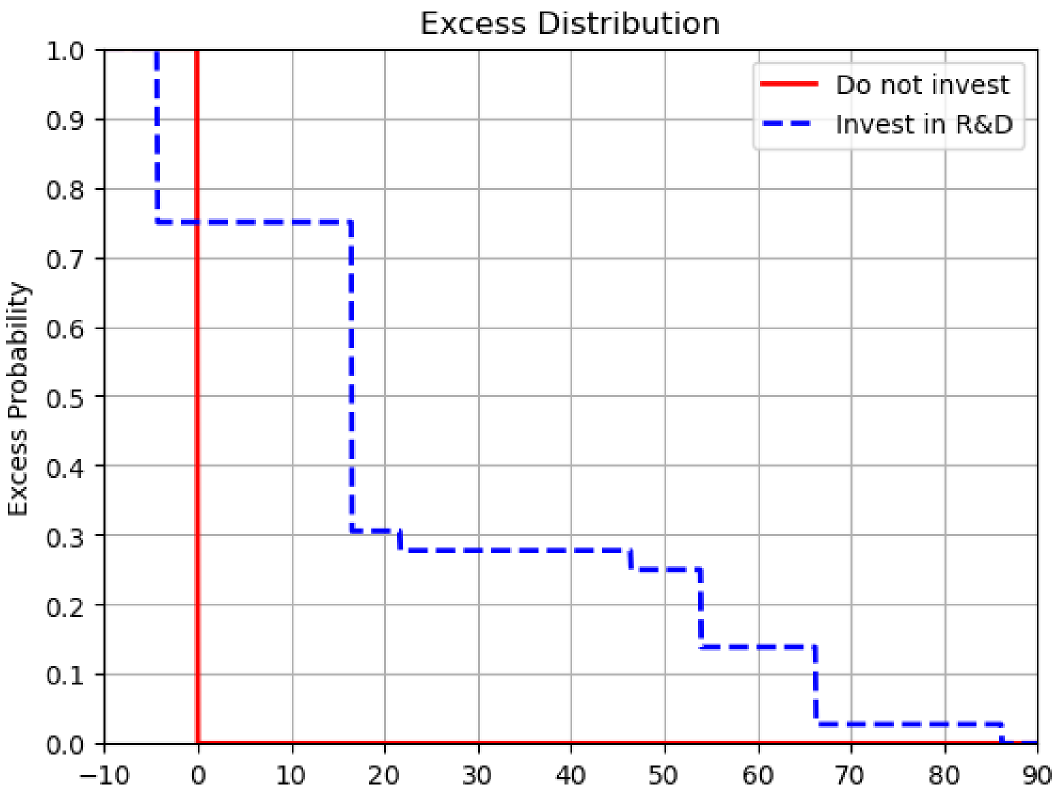
```
[6]: # Check for 1SD using is1SD function
compare_range = plot_range[:-1]
print("\nChecking for 1st Order Stochastic Dominances:")
for A, B in [(No_invest, Invest)]:
    if is1SD(A, B, compare_range, npoints):
        print(f"  {A.name} 1SD {B.name}")
    else:
        print(f"  {A.name} Does Not 1SD {B.name}")
```

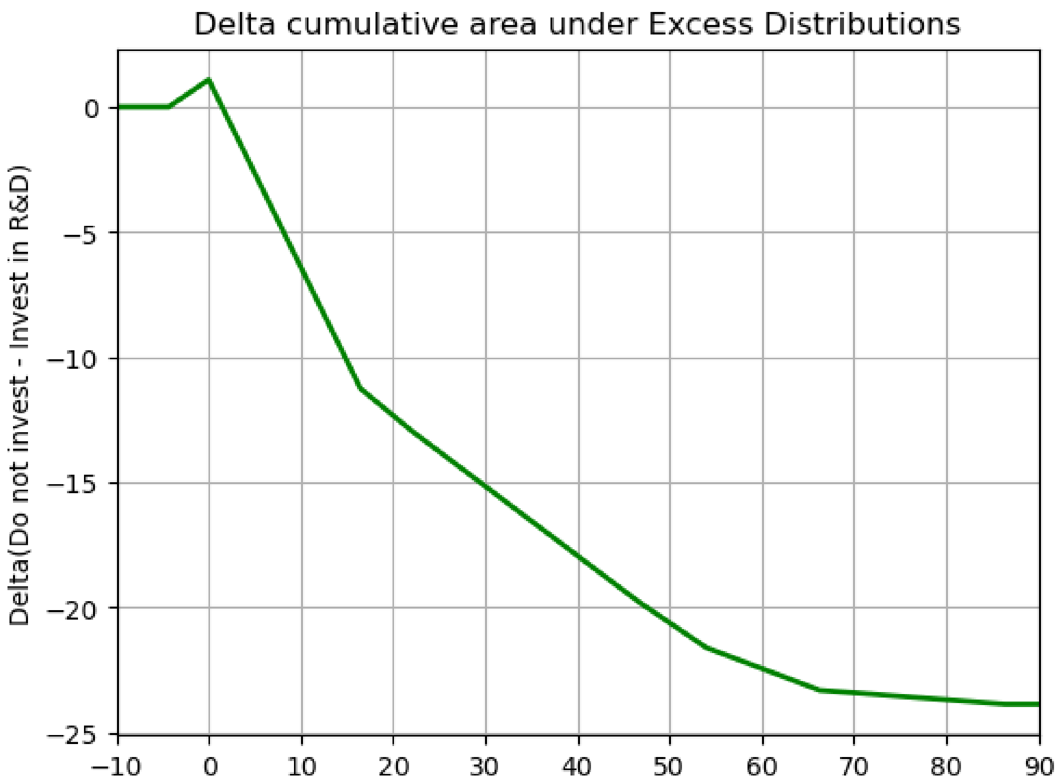
```
Checking for 1st Order Stochastic Dominances:
  Do not invest Does Not 1SD Invest in R&D
```

```
[7]: # Check for 2SD using is2SD function
for A, B in [(No_invest, Invest)]:
    print(f"\nChecking if {A.name} 2SD {B.name}:")
    if is2SD(A, B, plot_range, npoints, show_plot=True, dpi=100):
        print(f"\n{A.name} 2SD {B.name}")
    else:
        print(f"\n{A.name} Does Not 2SD {B.name}")
```

```
Checking if Do not invest 2SD Invest in R&D:
```




Excess Distribution
Excess Probability
Do not invest
Invest in R&D
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10
0
10
20
30
40
50
60
70
80
90


Delta cumulative area under Excess Distributions
Delta(Do not invest - Invest in R&D)
0
−5
−10
−15
−20
−25
−10
0
10
20
30
40
50
60
70
80
90

Do not invest Does Not 2SD Invest in R&D

[ ]:

```

```

2.3.3 LIM Problem: Stochastic Dominance Analysis

Source: 8.4.2_LIM_Stochastic_Dominance_Analysis.ipynb

[1]:

```python
""" LIM Problem: Stochastic Dominance Analysis """
from DecisionAnalysisPy import RiskDeal
from DecisionAnalysisPy import plot_risk_profiles, is1SD, is2SD
```

[2]:

```python
# Crate the risky deals
Train = RiskDeal(
    x=[ 120.264,   150.786,  171.025, 187.633, 214.429,
        230.102,   266.828,  288.499, 358.999 ],
    p=[ 0.095070, 0.137191, 0.191698, 0.042003, 0.276629,
        0.059869, 0.084695, 0.086394, 0.026451 ],
    name = 'Train User')

IPX = RiskDeal(
    x = [  115.809,  253.293,  315.189 ],
    p = [ 0.346637, 0.500215, 0.153149 ],
    name = 'Use IPX')
```

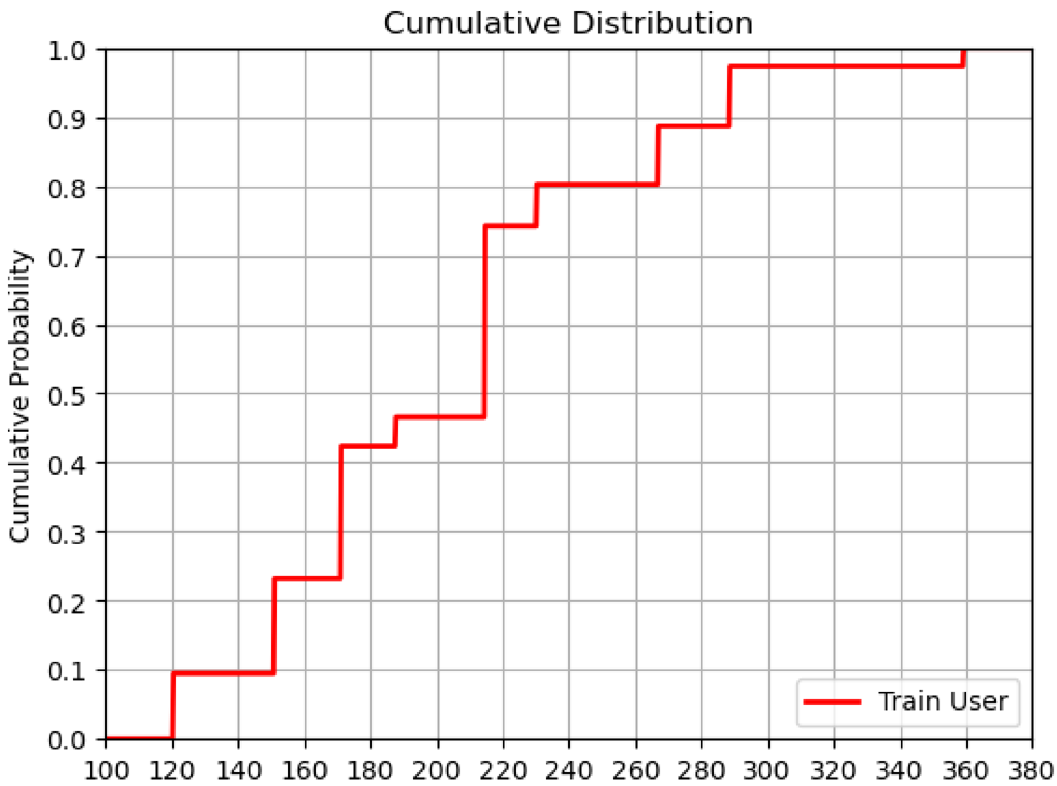
[3]:

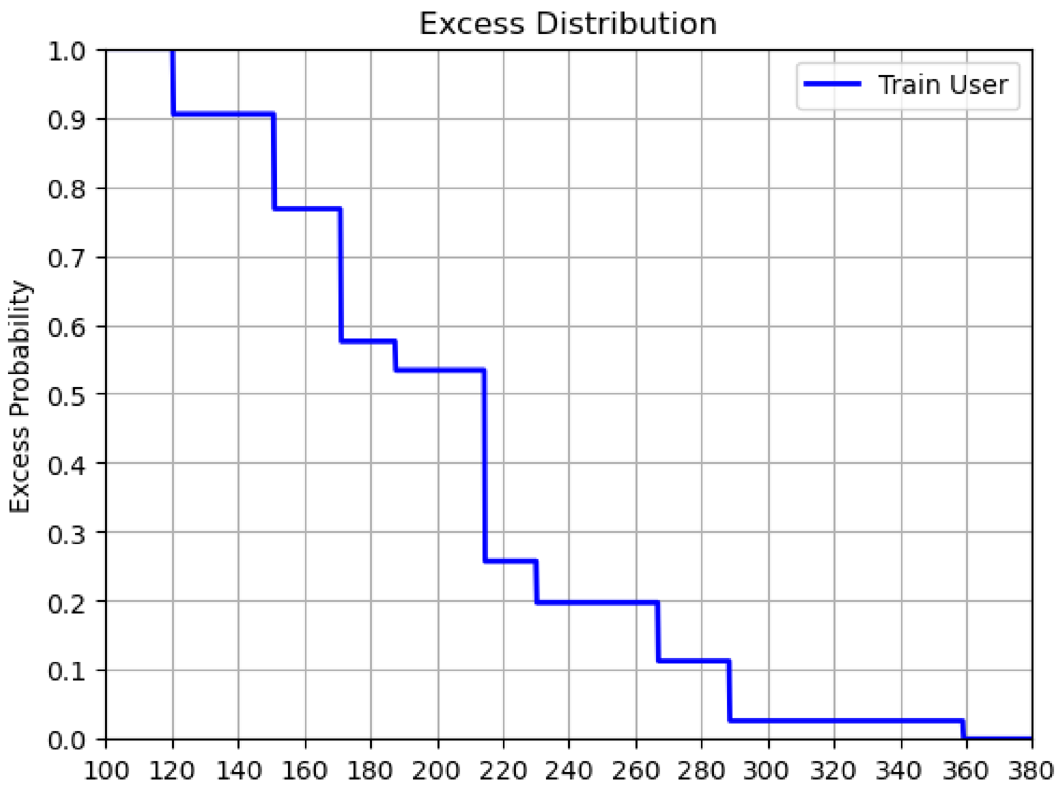
```python
# Parameters for plotting and stochastic dominance analysis
plot_range = (100, 380, 20)
npoints = 1000
```

[4]:

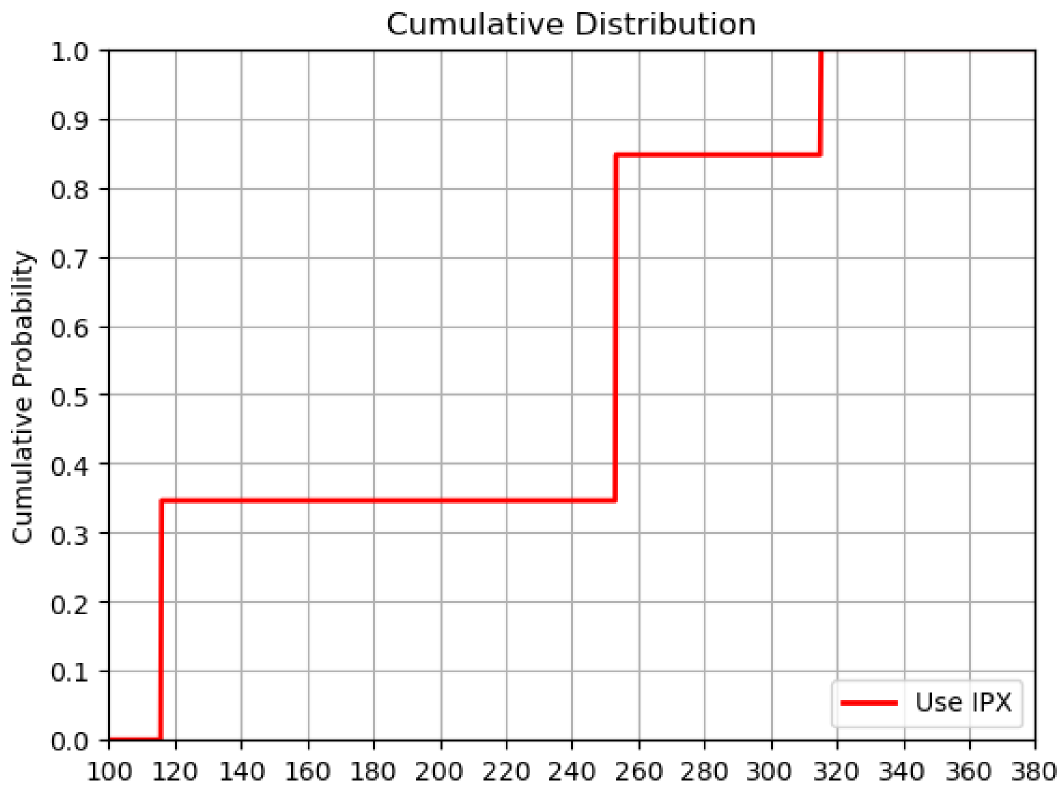
```python
# Plot the invidvual risk profiles
for deal in [Train, IPX]:
    print(f"\nRisk Profiles for {deal.name}:")
    deal.plot_CDF(plot_range, npoints, dpi=100)
    deal.plot_EPF(plot_range, npoints, dpi=100)
```

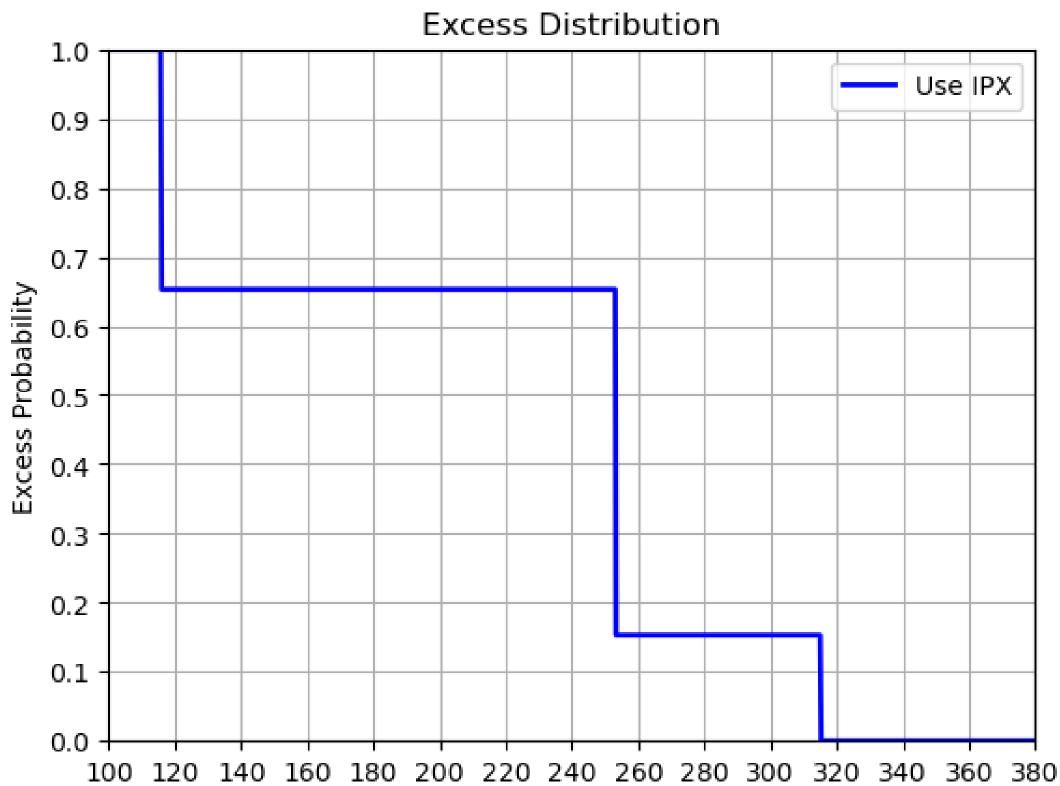
Risk Profiles for Train User:


Cumulative Distribution
Cumulative Probability
Train User
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
100 120 140 160 180 200 220 240 260 280 300 320 340 360 380


Excess Distribution
Excess Probability
Train User
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
100 120 140 160 180 200 220 240 260 280 300 320 340 360 380

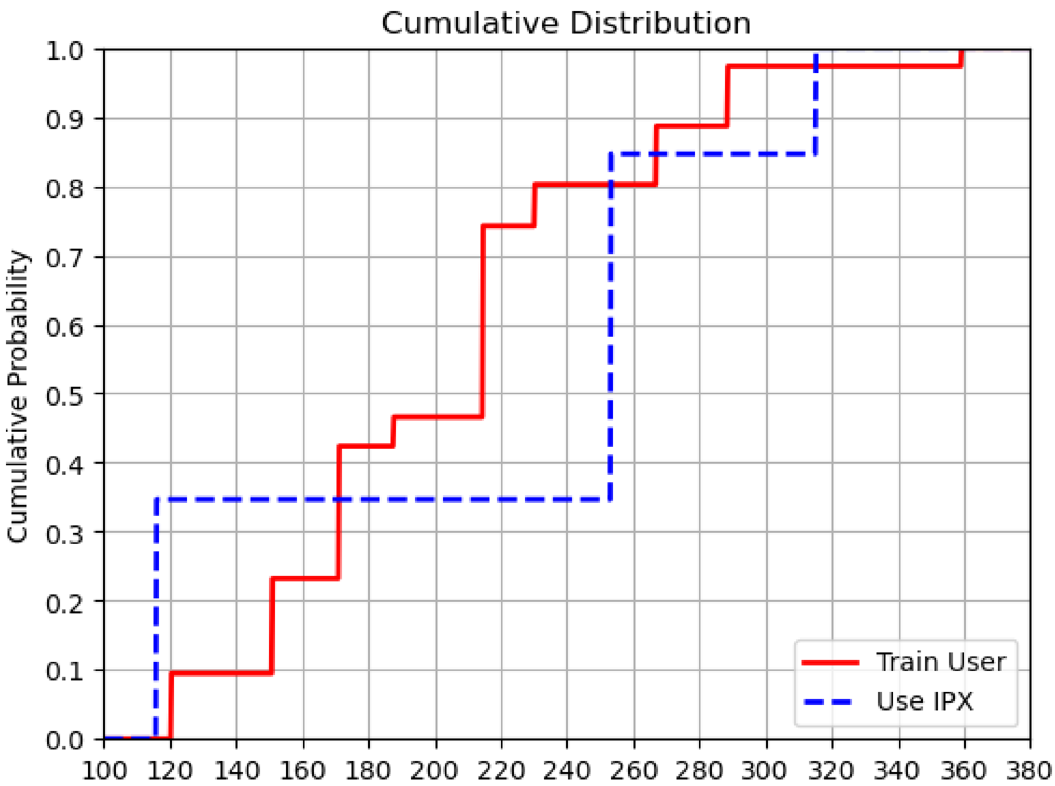
Risk Profiles for Use IPX:

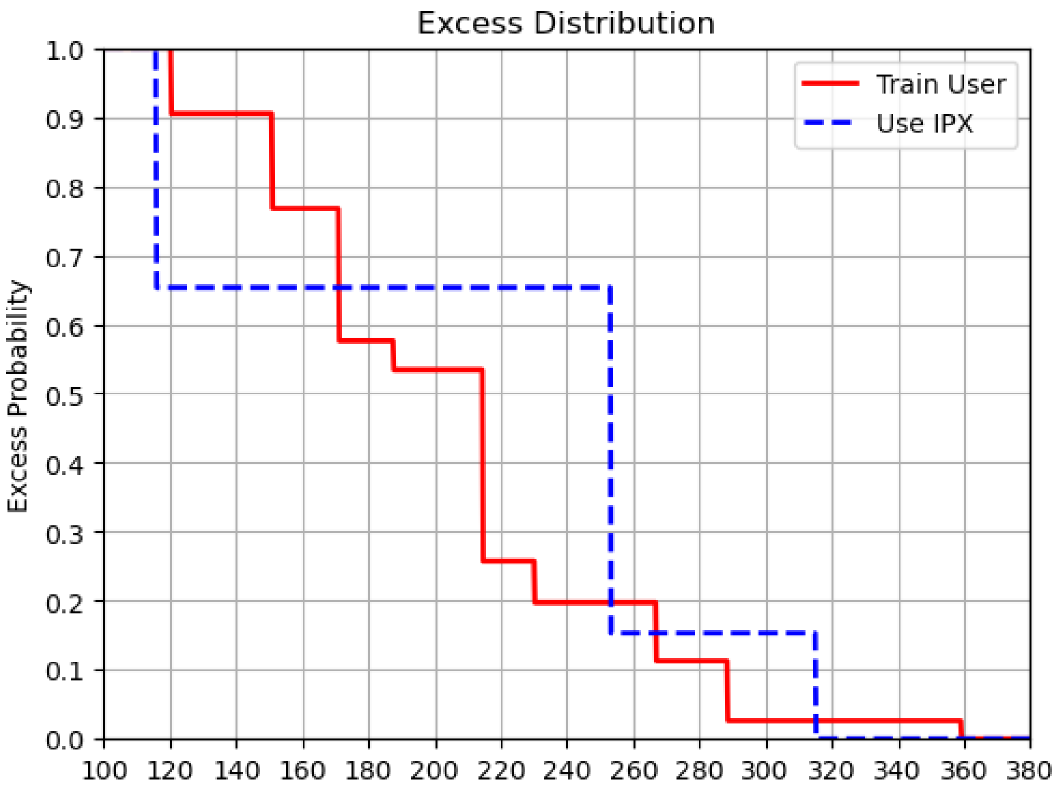




[5]:
```
# Plot the 2 risk profiles together for comparison
print("\nRisk Profiles for LIM Problem")
plot_risk_profiles([Train, IPX], plot_range, num=npoints,
                            CDF=True, EPF=True, dpi=100)
```

```
Risk Profiles for LIM Problem
```



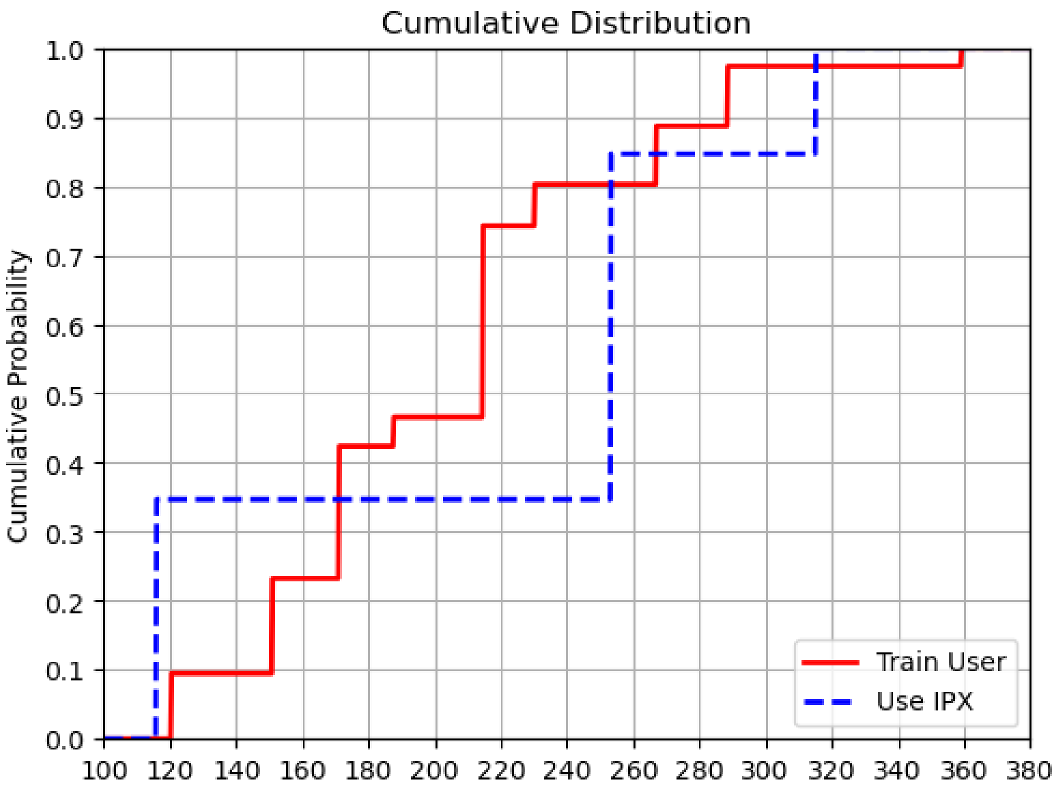


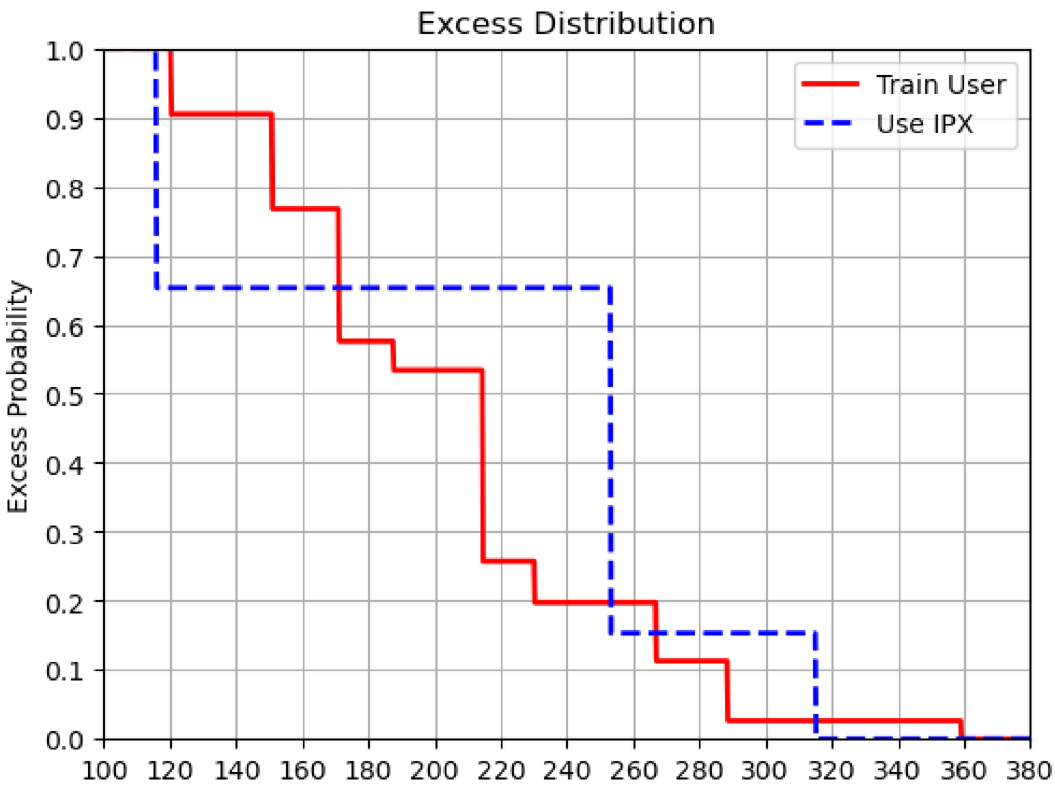
[6]:
```
# We only need to check if "Train User" dominates "IPX".
# Check for 1SD using is1SD function
compare_range = plot_range[:-1]
print("\nChecking for 1st Order Stochastic Dominances:")
for A, B in [(Train, IPX)]:
    if is1SD(A, B, compare_range, npoints):
        print(f"  {A.name} 1SD {B.name}")
    else:
        print(f"  {A.name} Does Not 1SD {B.name}")
```

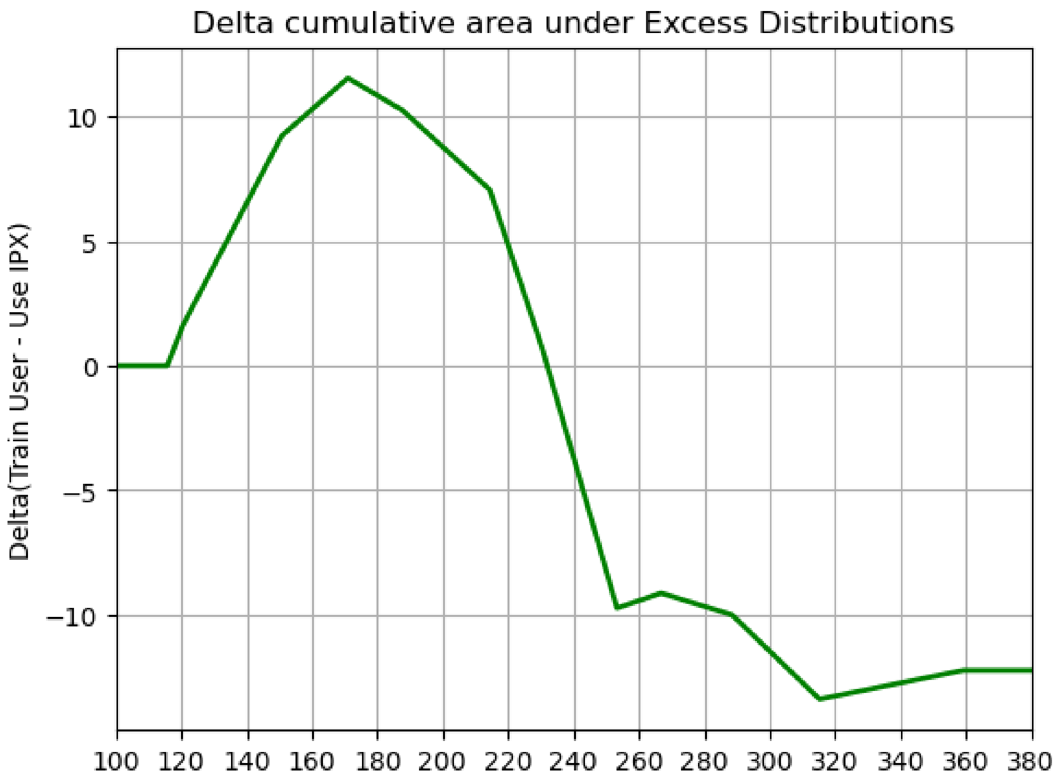
```
Checking for 1st Order Stochastic Dominances:
  Train User Does Not 1SD Use IPX
```

[7]:
```
# Check for 2SD using is2SD function
for A, B in [(Train, IPX)]:
    print(f"\nChecking if {A.name} 2SD {B.name}:")
    if is2SD(A, B, plot_range, npoints, show_plot=True, dpi=100):
        print(f"\n{A.name} 2SD {B.name}")
    else:
        print(f"\n{A.name} Does Not 2SD {B.name}")
```

```
Checking if Train User 2SD Use IPX:
```


Cumulative Distribution
Cumulative Probability
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
100
120
140
160
180
200
220
240
260
280
300
320
340
360
380
Train User
Use IPX


Excess Distribution
Excess Probability
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
100
120
140
160
180
200
220
240
260
280
300
320
340
360
380
Train User
Use IPX




```
Train User Does Not 2SD Use IPX
```

[ ]:

```

```

2.4 Personal Indifferent Buying and Selling Prices Computations

2.4.1 Personal Indifferent Selling Price under General Risk Preference

Source: 6.1.2_Selling_Price_under_General_Risk_Preference_using_RiskDeal_Class.ipynb

[1]:

```
""" Compute Personal Indifferent Selling Price (PISP) under
    General Risk Preference usikng RiskDeal Class """
from DecisionAnalysisPy import RiskDeal
import numpy as np
```

[2]:

```
# Create the risky deal
D61 = RiskDeal(x=[10, -5], p=[0.7, 0.3],
                states=['Win','Lose'], name='Example_6.1')
```

[3]:

```
# Define the wealth utility function
def uw(w):
    if w >= 0:
        return np.sqrt(w)
    else:
```

```python
        return -np.sqrt(-w)

# Define the wealth utility inverese function
def uw_inv(y):
    if y >= 0:
        return np.square(y)
    else:
        return -np.square(y)
```

[4]:
```python
# # Compute the PISP at different w0 with and without invese function
for w0 in [10, 100, 1000, 10000]:
    print(f"\nInitial wealth = {w0}")

    # Compute PISP with inverse wealth utility function provided
    s1 = D61.PISP(w0, uw, uw_inv)
    print(f"  Selling price  = {s1:.6f}")

    # Compute PISP without inverse wealth utility function
    s2 = D61.PISP(w0, uw)
    print(f"  Selling price  = {s2:.6f}")

    # Risk preimum is EV - PISP
    pi = D61.EV - s1
    print(f"  Risk Premium   ={pi:.6f}")
```

```
Initial wealth = 10
  ..using inverse wealth utility function to find PISP
  Selling price  = 4.450000
  ..using solver hybr to find PISP
  Selling price  = 4.450000
  Risk Premium   =1.050000

Initial wealth = 100
  ..using inverse wealth utility function to find PISP
  Selling price  = 5.384601
  ..using solver hybr to find PISP
  Selling price  = 5.384601
  Risk Premium   =0.115399

Initial wealth = 1000
  ..using inverse wealth utility function to find PISP
  Selling price  = 5.488217
  ..using solver hybr to find PISP
  Selling price  = 5.488217
  Risk Premium   =0.011783

Initial wealth = 10000
```

```
  ..using inverse wealth utility function to find PISP
  Selling price  = 5.498819
  ..using solver hybr to find PISP
  Selling price  = 5.498819
  Risk Premium   =0.001181
```

[ ]:

2.4.2 Personal Indifference Buying Price under General Risk Preference

Source: 6.1.3_Buying_Price_under_General_Risk_Preference_using_RiskDeal_Class.ipynb

[1]:
```python
""" Compute Personal Indifferent Buying Price under
    General Risk Preference using RiskDeal Class """
from DecisionAnalysisPy import RiskDeal
import numpy as np
```

[2]:
```python
# Create the risky deal
D61 = RiskDeal(x=[10, -5], p=[0.7, 0.3], states=['Win','Lose'],
               name='Example_6.1')
```

[3]:
```python
# Define the wealth utility function. Inverse not needed.
def uw(w):
    if w >= 0:
        return np.sqrt(w)
    else:
        return -np.sqrt(-w)
```

[4]:
```python
# Compute the personal indifferent buying price at different w0
for w0 in [10, 100, 1000, 10000]:
    print(f"\nInitial wealth = {w0}")

    # Compute PIBP with default solver
    b1 = D61.PIBP(w0, uw)
    print(f"Buying price   = {b1:.6f}")

    # Compute PIBP with lm solver
    b2 = D61.PIBP(w0, uw, method='lm')
    print(f"Buying price   = {b2:.6f}")
```

```
Initial wealth = 10
  ..using solver hybr to find PIBP
Buying price   = 3.726958
  ..using solver lm to find PIBP
Buying price   = 3.726958

Initial wealth = 100
  ..using solver hybr to find PIBP
```

```
Buying price    = 5.378193
  ..using solver lm to find PIBP
Buying price    = 5.378193

Initial wealth = 1000
  ..using solver hybr to find PIBP
  ..message: The iteration is not making good progress, as measured by␣
↪the
  improvement from the last ten iterations.
  ..number of function evaluations: 17
Buying price    = 5.488152
  ..using solver lm to find PIBP
Buying price    = 5.488152

Initial wealth = 10000
  ..using solver hybr to find PIBP
Buying price    = 5.498818
  ..using solver lm to find PIBP
Buying price    = 5.498818
```

[ ]:

```

```

2.4.3 PISP Independent of Wealth

Source: 6.2.4_Selling_Price_indep_Wealth_under_Delta_Property_RiskDeal_Class.ipynb

[1]:

```
""" Demostrate Selling Price independent of Wealth under Delta Property␣
↪"""
from DecisionAnalysisPy import RiskDeal
import numpy as np
```

[2]:

```
# Create the risky deal
Deal = RiskDeal(x=[100, 0], p=[0.5, 0.5], states=['up','down'],
                name='Coin_game_0_100')
```

[3]:

```
# Define the wealth utility function and its inverse
uw = lambda w : 1 - 2**(-w/50)
uw_inv = lambda y : -50*np.log(1-y)/np.log(2)
```

[4]:

```
# Compute PISP at different initial wealth
for w0 in [0, 100, 500, 1000]:
    print(f"\nInitial wealth = {w0}")

    # Compute PISP with inverse wealth utility function provided
    s1 = Deal.PISP(w0, uw, uw_inv)
    print(f"Selling price  = {s1:.6f}")

    # Compute PISP without inverse wealth utility function
```

```
    s2 = Deal.PISP(w0, uw)
    print(f"Selling price  = {s2:.6f}")
```

```
Initial wealth = 0
  ..using inverse wealth utility function to find PISP
Selling price  = 33.903595
  ..using solver hybr to find PISP
Selling price  = 33.903595

Initial wealth = 100
  ..using inverse wealth utility function to find PISP
Selling price  = 33.903595
  ..using solver hybr to find PISP
Selling price  = 33.903595

Initial wealth = 500
  ..using inverse wealth utility function to find PISP
Selling price  = 33.903595
  ..using solver hybr to find PISP
Selling price  = 33.903595

Initial wealth = 1000
  ..using inverse wealth utility function to find PISP
Selling price  = 33.903595
  ..using solver hybr to find PISP
Selling price  = 33.903595
```

[ ]:

```
```

2.4.4 PISP = PIBP under Delta Property

Source: 6.2.5_Buying_Price_eq_Selling_Price_under_Delta_Property_RiskDeal_Class.ipynb

[1]:

```
""" Demostrate Buying Price = Selling Price under Delta Property """
from DecisionAnalysisPy import RiskDeal
import numpy as np
```

[2]:

```
# Define the exponential wealth utility function and its inverse
uw = lambda w : 1 - 2**(-w/50)
uw_inv = lambda y : -50*np.log(1-y)/np.log(2)
```

[3]:

```
# Fix the initial wealth
w0 = 500
# Create the base risky deal
base_deal = RiskDeal(x=[10, 0], p=[0.5,0.5], states=['up','down'])
```

[4]:
```python
# Compute selling and buying price of base deal at w0=500
s0 = base_deal.PISP(w0, uw, uw_inv)
print(f"Selling price of base deal = {s0:.6f}")
b0 = base_deal.PIBP(w0, uw)
print(f"Buying price of base deal  = {b0:.6f}")
print("Notice that PISP = PIBP?")
```

```
  ..using inverse wealth utility function to find PISP
Selling price of base deal = 4.826852
  ..using solver hybr to find PIBP
Buying price of base deal  = 4.826852
Notice that PISP = PIBP?
```

[5]:
```python
# Compute PISP, PIBP under different delta shifts
for delta in [-20, 20, 50, 100]:
    print(f"\nDelta = {delta}:")
    shifted_deal = RiskDeal(x=[10+delta, delta], p=[0.5, 0.5],
                            states=['up','down'])
    s1 = shifted_deal.PISP(w0, uw, uw_inv)
    # s1 = Deal.PISP(w0, wuf) # use solver instead
    print(f"  Selling price of shifted deal  = {s1:.6f}")
    print(f"    Shift in selling price = {s1-s0:.6f}")
    b1 = shifted_deal.PIBP(w0, uw)
    print(f"  Buying price    = {b1:.6f}")
    print(f"    Shift in buying price = {b1-b0:.6f}")

print("\nNotice that Selling Price = Buying Price under all delta␣
 ↪shifts")
print("  and they are also all shifted by delta")
```

```

Delta = -20:
  ..using inverse wealth utility function to find PISP
  Selling price of shifted deal  = -15.173148
    Shift in selling price = -20.000000
  ..using solver hybr to find PIBP
  Buying price    = -15.173148
    Shift in buying price = -20.000000

Delta = 20:
  ..using inverse wealth utility function to find PISP
  Selling price of shifted deal  = 24.826852
    Shift in selling price = 20.000000
  ..using solver hybr to find PIBP
  Buying price    = 24.826852
    Shift in buying price = 20.000000

Delta = 50:
  ..using inverse wealth utility function to find PISP
```

```
  Selling price of shifted deal  = 54.826852
    Shift in selling price = 50.000000
  ..using solver hybr to find PIBP
  Buying price   = 54.826852
    Shift in buying price = 50.000000

Delta = 100:
  ..using inverse wealth utility function to find PISP
  Selling price of shifted deal  = 104.826852
    Shift in selling price = 100.000000
  ..using solver hybr to find PIBP
  Buying price   = 104.826852
    Shift in buying price = 100.000000

Notice that Selling Price = Buying Price under all delta shifts
  and they are also all shifted by delta
```

[ ]:

```

```

2.4.5 Kim's PISP and PIBP for Coin Tossing Game

Source: 6.2.5_Kim_PISP_PIBP_for_CoinTossingGame_using_RiskDeal_Class.ipynb

[1]:

```
""" Kim's buying and selling prices for the $100 coin tossing game
    This example shows that Kim's PISP = PIBP for all w0 """
from DecisionAnalysisPy import RiskDeal
import numpy as np
```

[2]:

```
# Create the risky deal
coin_game = RiskDeal(x=[0, 100], p=[0.5,0.5], states=['lose','win'],
                     name='Coin_Tossing_Game_100')
```

[3]:

```
# Define the utility function and its inverse
RT = 50/np.log(2)
uw   = lambda w: (4/3)*(1 - np.exp(-w/RT))
uw_inv = lambda y: -RT*np.log(1-3*y/4)

# Compute buying price, selling price, and riks preimum at different w0
for w0 in [10, 100, 1000]:
    print(f"\nInitial wealth = {w0}")

    s1 = coin_game.PISP(w0, uw, uw_inv)
    print(f"  Selling price  = {s1:.6f}")

    b1 = coin_game.PIBP(w0, uw)
    print(f"  Buying price   = {b1:.6f}")

    pi = ev = coin_game.EV - s1
```

```python
    print(f"  Risk Premium    = {pi:.6f}")

print("Notice that PISP = PIBP, and they are independent of w0?")
```

```
Initial wealth = 10
  ..using inverse wealth utility function to find PISP
  Selling price  = 33.903595
  ..using solver hybr to find PIBP
  Buying price   = 33.903595
  Risk Premium   = 16.096405

Initial wealth = 100
  ..using inverse wealth utility function to find PISP
  Selling price  = 33.903595
  ..using solver hybr to find PIBP
  Buying price   = 33.903595
  Risk Premium   = 16.096405

Initial wealth = 1000
  ..using inverse wealth utility function to find PISP
  Selling price  = 33.903595
  ..using solver hybr to find PIBP
  Buying price   = 33.903595
  Risk Premium   = 16.096405
Notice that PISP = PIBP, and they are independent of w0?
```

[ ]:

```python

```

3 Class ExpUtilityFunction

3.1 Documentation

[1]:
```
from DecisionAnalysisPy import ExpUtilityFunction
```

[2]:
```
print(ExpUtilityFunction.__doc__)
```

```
Class for Exponential Utility Function u(x)= a - b exp(-x/RT)
    ExpUtilityFunction(RT=None, L=None, H=None):
    Parameters:
      RT = risk tolerance. If omitted, can be determined later
      L = lower bound such that u(L) = 0 (optional)
      H = Upper boundary such that u(H) = 1 (optional)

    Attributes:
      RT = risk tolerance
      L = lower bound such that u(L) = 0
      H = upper bound such that u(H) = 1
      a = constant that fits the boundary conditions u(L)=0, u(H)=1
      b = constant that fits the boundary conditions u(L)=0, u(H)=1
               b > 0 if RT > 0,  b < 0 if RT < 0
      risk_attitude = "risk averse" or "risk seeking"

    Methods:
      set_RT(RT):  Set risk tolerance to RT without changing L and H
      set_bounds(L, H):  Set L and H without changing RT
      u(x): Compute and return u(x)
      find_RT_5050CE(L, H, X05, guess): Find RT using 50-50 CE method
      plot(xmin, xmax): Plot the utility function from xmin(L) to xmax(H)
      fun_str(): Return a string representation of the utility function
```

[ ]:
```

```

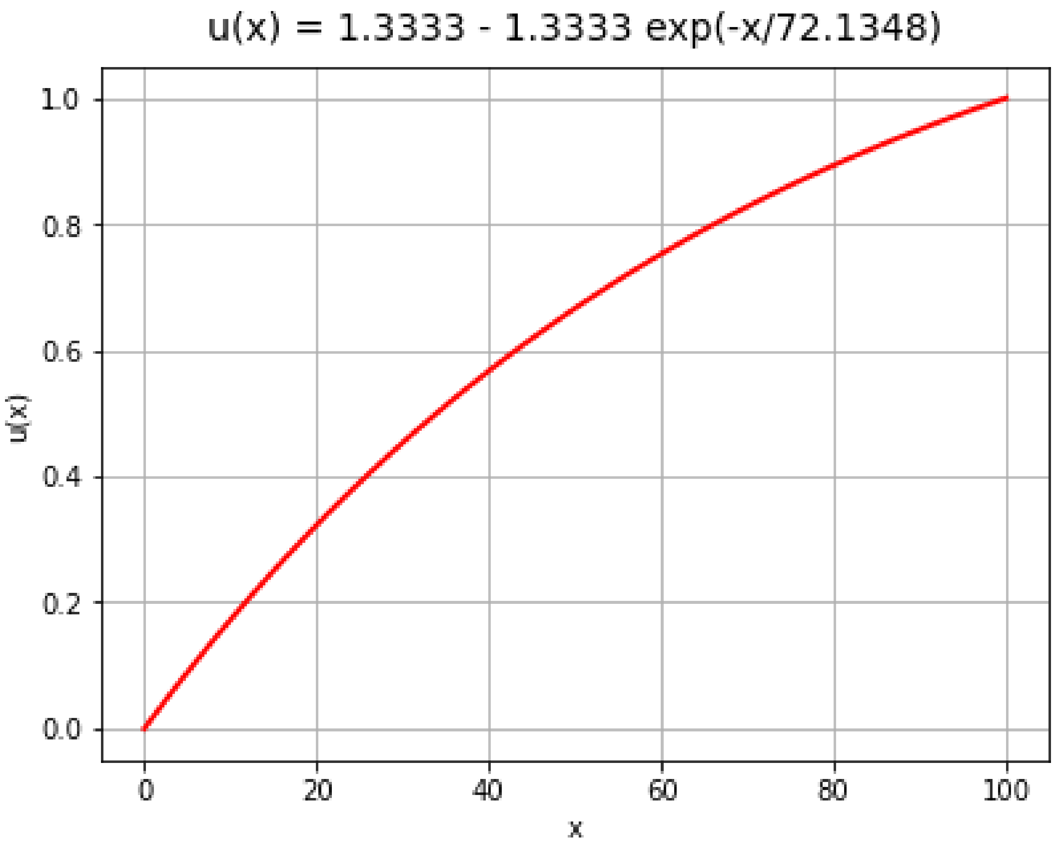
3.2 Plot Kim's Utility Function

Source: 6.3.1_Plot_Kim_Utility_Function_using_ExpUtilityFunction_Class.ipynb

[1]:
```
""" Fit Kim's utility function to boundary conditions and plot it
    using ExpUtilityFunction method """
from DecisionAnalysisPy import ExpUtilityFunction
import numpy as np
```

[2]:
```
# Create and plot Kim's utility function directly using L, H and RT
RT = 50/np.log(2)
f1 = ExpUtilityFunction(L=0, H=100, RT=RT)
```

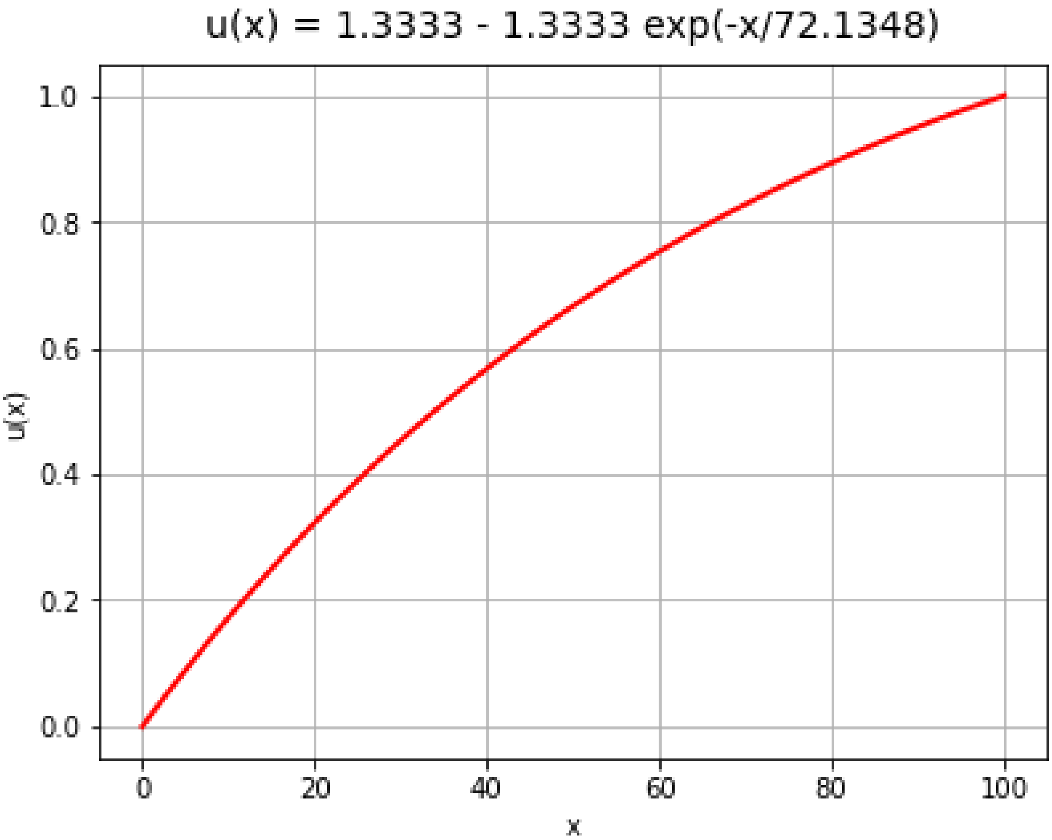
[3]:
```
# Plot it and display key parameters
f1.plot()
print(f"  Utility function is {f1.fun_str()}")
print(f"  Parameters = {f1.params()}")
print("Check some utility values:")
print(f"  u(0) = {f1.u(0)}")
print(f"  u(50) = {f1.u(50)}")
print(f"  u(100) = {f1.u(100)}")
```




```
  Utility function is u(x) = 1.3333 - 1.3333 exp(-x/72.1348)
  Parameters = {'L': 0, 'H': 100, 'RT': 72.13475204444818, 'a':
1.3333333333333333, 'b': 1.3333333333333333, 'risk_attitude': 'risk␣
 ↪averse'}
Check some utility values:
  u(0) = 0.0
  u(50) = 0.6666666666666666
  u(100) = 1.0
```

[4]:
```
# Create an instance of the function first and then set the parameters
f2 = ExpUtilityFunction()
f2.set_bounds(0, 100)
f2.set_RT(RT)
```

```
f2.plot()
print(f"  Utility function is {f2.fun_str()}")
print(f"  Parameters = {f2.params()}")
print("Check some utility values:")
print(f"  u(0) = {f2.u(0)}")
print(f"  u(50) = {f2.u(50)}")
print(f"  u(100) = {f2.u(100)}")
```




```
  Utility function is u(x) = 1.3333 - 1.3333 exp(-x/72.1348)
  Parameters = {'L': 0, 'H': 100, 'RT': 72.13475204444818, 'a':
1.3333333333333333, 'b': 1.3333333333333333, 'risk_attitude': 'risk␣
↪averse'}
Check some utility values:
  u(0) = 0.0
  u(50) = 0.6666666666666666
  u(100) = 1.0
```

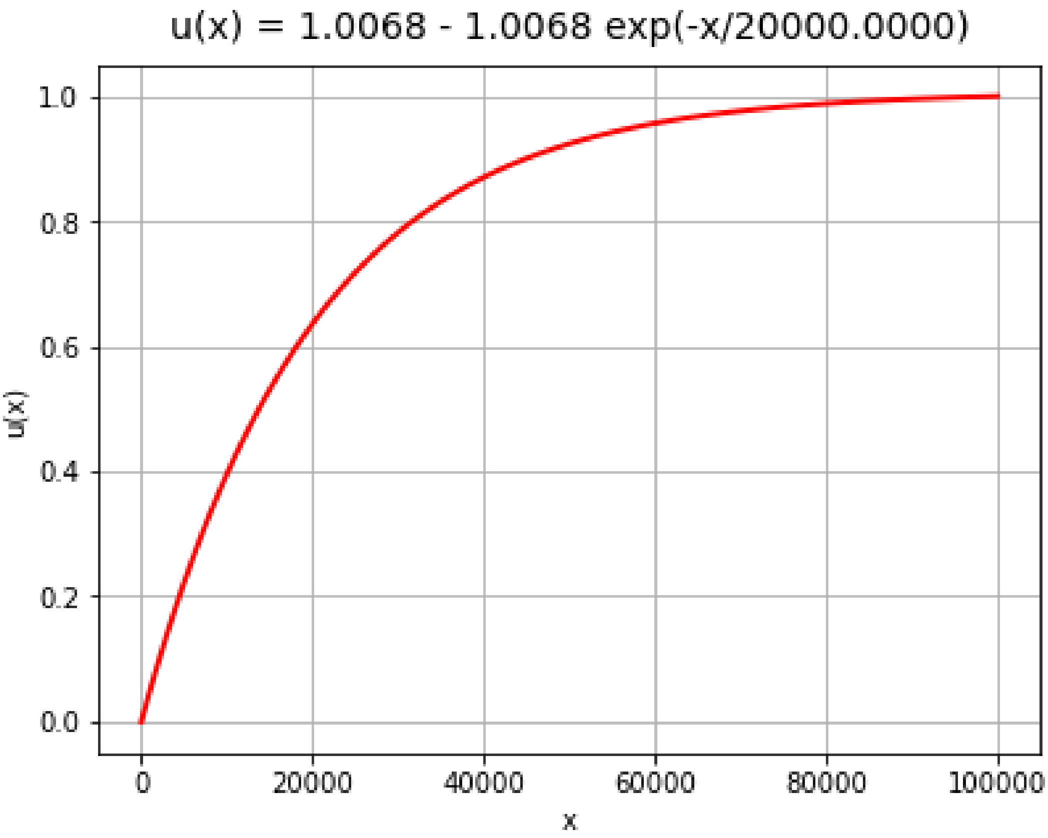
```
[ ]:
```

3.3 Plot Exponential Utility functions

Source: 6.3.2_Plot_Exponential_Utility_Functions_using_ExpUtilityFunction_Class.ipynb

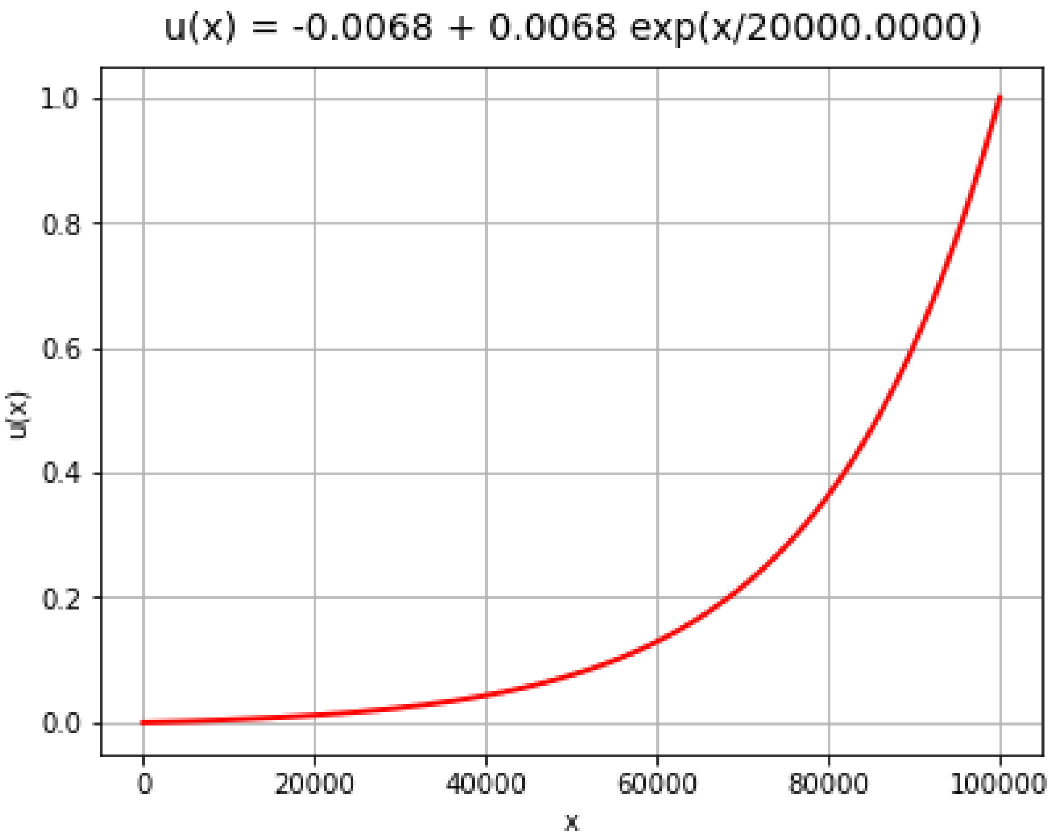
[1]:
```
""" Plot Exponential Utility Functions using ExpUtilityFunction Class␣
↪"""
from DecisionAnalysisPy import ExpUtilityFunction
```

[2]:
```
RT1 = 20000
f1 = ExpUtilityFunction(L=0, H=100000, RT=RT1)
f1.plot()
print(f"  Utility function is {f1.fun_str()}")
print(f"  Parameters = {f1.params()}")
```




```
  Utility function is u(x) = 1.0068 - 1.0068 exp(-x/20000.0000)
  Parameters = {'L': 0, 'H': 100000, 'RT': 20000, 'a': 1.
↪0067836549063043, 'b':
1.0067836549063043, 'risk_attitude': 'risk averse'}
```

[3]:
```
RT2 = -20000
f2 = ExpUtilityFunction(L=0, H=100000, RT=RT2)
f2.plot()
print(f"  Utility function is {f2.fun_str()}")
print(f"  Parameters = {f2.params()}")
```




```
Utility function is u(x) = -0.0068 + 0.0068 exp(x/20000.0000)
Parameters = {'L': 0, 'H': 100000, 'RT': -20000, 'a': -0.
↪006783654906304231,
'b': -0.006783654906304231, 'risk_attitude': 'risk seeking'}
```

[ ]:

3.4 Find RT using 50-50 CE Method

Source: 6.3.3_Find_RT_using_5050CE_method_with_ExpUtilityFunction_Class.ipynb

[1]:
```
""" Find Risk Tolerance using 50-50 CE Method with ExpUtilityFunction␣
↪Class """
from DecisionAnalysisPy import ExpUtilityFunction
import numpy as np
from scipy.optimize import root
```

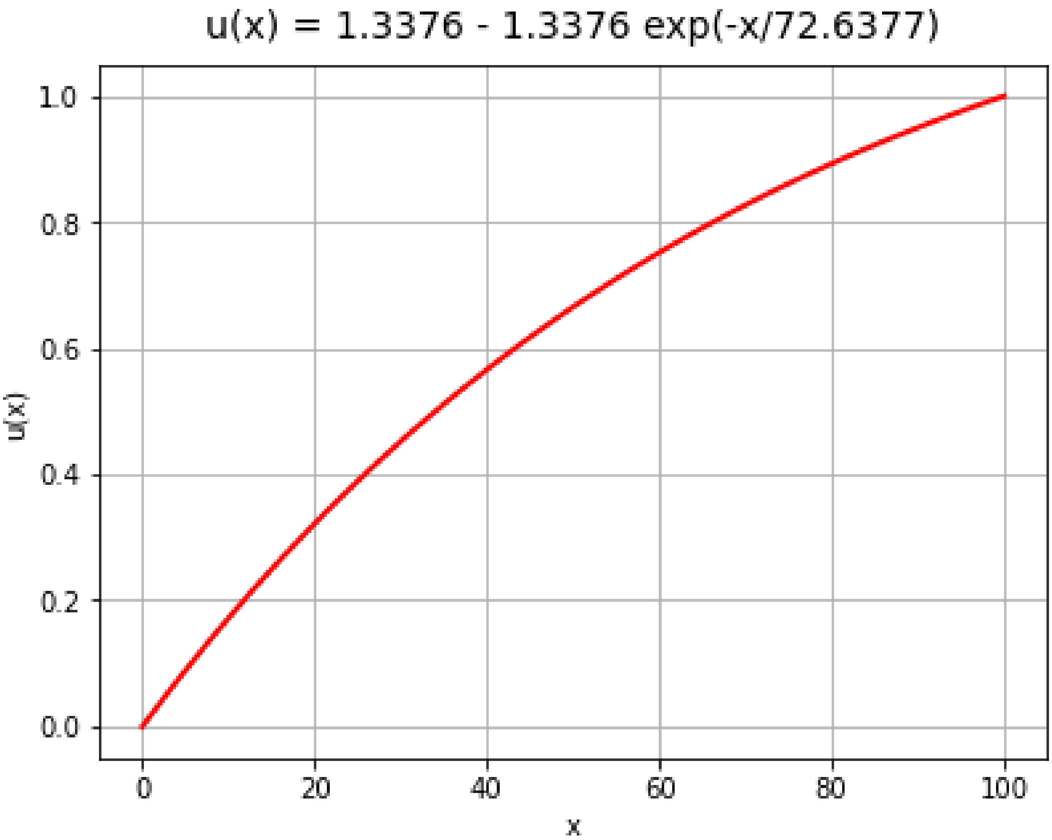
[2]:
```
# Case 1: Risk Averse decision maker
print("\nRisk averse case:")
L, H, X05 = 0, 100, 34
```

```
Risk averse case:
```

[3]:
```python
# First, let's do it the hard way by showing off your solver skills.
eq=lambda r: (1.0-np.exp(-(X05-L)/r))/(1.0-np.exp(-(H-L)/r))-0.5
sol = root(eq, x0=50)
if not sol.success:print("Warning:", sol.message)
RT = sol.x[0]
print(f"Risk Tolerence = {RT}")
```

```
Risk Tolerence = 72.63770314222594
```

[4]:
```python
# Next, do it the easy way using ExpUtilityFunction.find_RT_5050CE␣
↪method
# Create an "empty" function and then find RT using 50-50 CE method
f1 = ExpUtilityFunction()
rt1 = f1.find_RT_5050CE(L, H, X05, guess=50)
f1.plot()

print(f"  Risk tolerance = {rt1}")
print(f"  Utility function is {f1.fun_str()}")
print(f"  Parameters = {f1.params()}")
print( "  Check key utility values:")
print(f"    u({L}) = {f1.u(L)}")
print(f"    u({X05}) = {f1.u(X05)}")
print(f"    u({H}) = {f1.u(H)}")
```



```
  Risk tolerance = 72.63770314222594
  Utility function is u(x) = 1.3376 - 1.3376 exp(-x/72.6377)
  Parameters = {'L': 0, 'H': 100, 'RT': 72.63770314222594, 'a':
1.337633855020259, 'b': 1.337633855020259, 'risk_attitude': 'Risk␣
↪averse'}
  Check key utility values:
    u(0) = 0.0
    u(34) = 0.5
    u(100) = 1.0
```

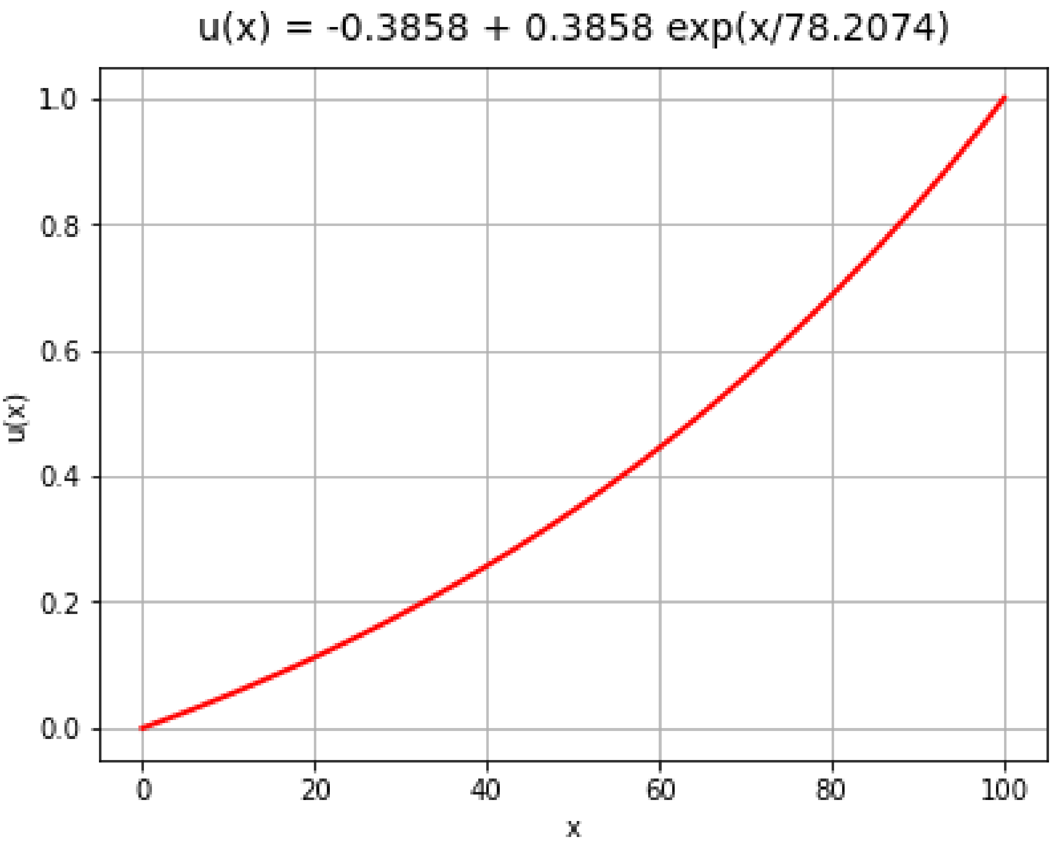
[5]:
```
# Case 2: Risk Seeking decision maker
print("\nRisk seeking case:")
L, H, X05 = 0, 100, 65
```

```
Risk seeking case:
```

[6]:
```
# First, let's do it the hard way by showing off your solver skills.
eq=lambda r: (1.0-np.exp(-(X05-L)/r))/(1.0-np.exp(-(H-L)/r))-0.5
sol = root(eq, x0=-50)
if not sol.success:print("Warning:", sol.message)
RT = sol.x[0]
print(f"Risk Tolerence = {RT}")
```

```
Risk Tolerence = -78.20735007721643
```

[7]:
```
 # Next, do it the easy way using ExpUtilityFunction.find_RT_5050CE␣
 ↪method
# Create an "empty" function and then find RT using 50-50 CE method
f2 = ExpUtilityFunction()
rt2 = f2.find_RT_5050CE(L, H, X05, guess=-50)
f2.plot()

print(f"  Risk tolerance = {rt2}")
print(f"  Utility function is {f2.fun_str()}")
print(f"  Parameters = {f2.params()}")
print( "  Check key utility values:")
print(f"    u({L}) = {f2.u(L)}")
print(f"    u({X05}) = {f2.u(X05)}")
print(f"    u({H}) = {f2.u(H)}")
```




```
  Risk tolerance = -78.20735007721643
  Utility function is u(x) = -0.3858 + 0.3858 exp(x/78.2074)
  Parameters = {'L': 0, 'H': 100, 'RT': -78.20735007721643, 'a':
-0.38583292058655677, 'b': -0.38583292058655677, 'risk_attitude': 'Risk
seeking'}
  Check key utility values:
    u(0) = 0.0
    u(65) = 0.49999999999999994
    u(100) = 1.0
```

[ ]:

4 Class DistFit_continuous

4.1 Documentation

[1]:
```
from DecisionAnalysisPy import DistFit_continuous
print(DistFit_continuous.__doc__)
```

```
Class for fitting Continuous Distrbutions to Data using
       scipy.stats.rv_continuous.fit

   DistFit(data):
     Parameter:
         data = 1-D Array of data to fit.
     Attributes:
         data = Data used for fitting
         results = DataFrame of fitted results
         Distributions = List of distributions to fit

     Methods:
         data_hist(bins=None, dpi=100):
             Plot a density histogram of the data
           Parameters:
             bins = number of bin to plot (default = None)
             dpi = dpi to plot (default=100)

         data_describe():
             Describe the data descriptive statistics

         fit(Dists, method, options):
             Fit the named distributions in the list
           Parameters:
             Dists = list of distributions to fit
             method = 'MLE' (default) or 'MM'
             options = dict of other options to pass to
                            scipy.stats.rv_continuous.fit

         plot_pdf(N=3, bins=None, dpi=100):
             Plot the PDFs of the fitted distributions over the data
           Parameters:
             N = number of top distributions to plot (default=3)
             bins = number of bins to plot data
             dpi = dpi to plot (default = 100)

         plot_cdf(N=3, dpi=100)
             Plot the CDFs of the fitted distributions over the data␣
↪ECDF
           Parameters:
             N = number of top distributions to plot (default=3)
```

```
        dpi = dpi to plot (default=100)

    parameters(N=3):
        Show parameters of top fitted results
      Parameters:
        N = number of top distributions to show (default=3)
      Return:
        Nested dictionary of fitted results.
```

[ ]:

```

```

4.2 Example 1: Fitting Continuous Functions to Data

Source: 7.4.3_Fit_Continuous_Distributions_to_Data_using_DistFit_continuous.ipynb

[1]:

```
""" Fit continuous distributions using DistFit_continuous Class """
from DecisionAnalysisPy import DistFit_continuous
import pandas as pd
```

[2]:

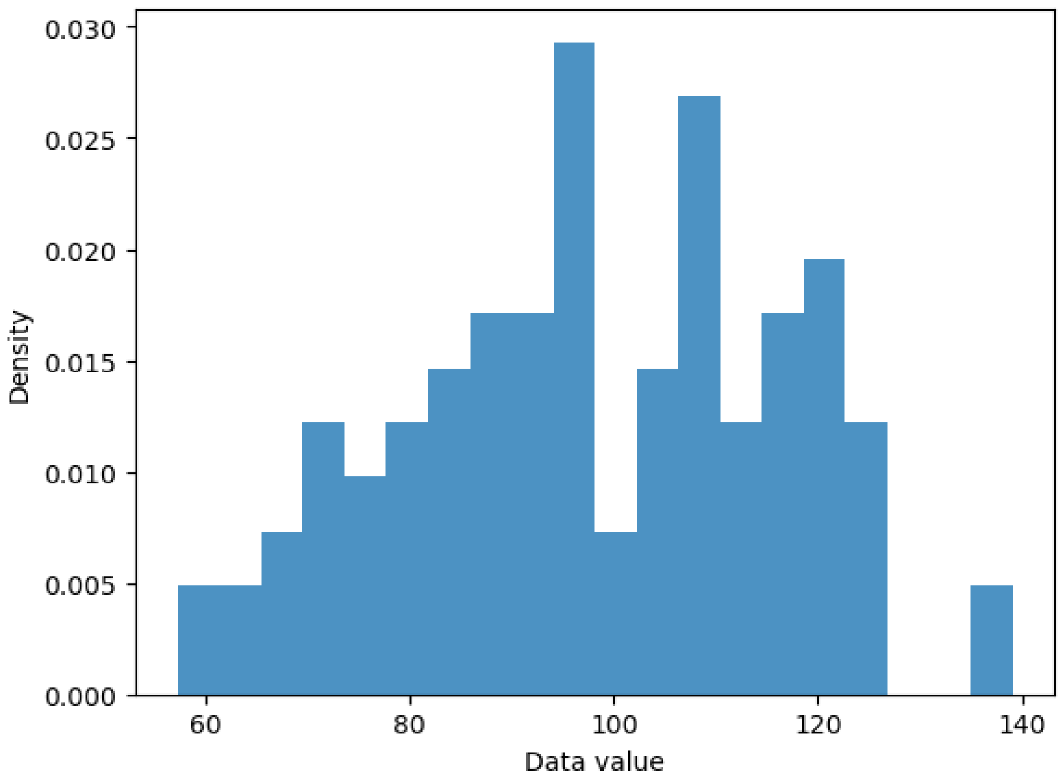
```
# Read the data from Excel file
data = pd.read_excel ("7.4.3_data.xlsx", sheet_name="data1",␣
 ↪header=None)
data = data.values.flatten()
```

[3]:

```
# List of distributions to fit
Dists = ['beta','expon','gamma','lognorm','logistic', 'laplace',
         'norm', 'rayleigh','triang','uniform','weibull_min']

# Use DisFit_continuous with the data
ex1 = DistFit_continuous(data)
```

[4]:

```
# Visualise and describe the data
bins = 20
ex1.data_hist(bins=bins)
results = ex1.data_describe()
```



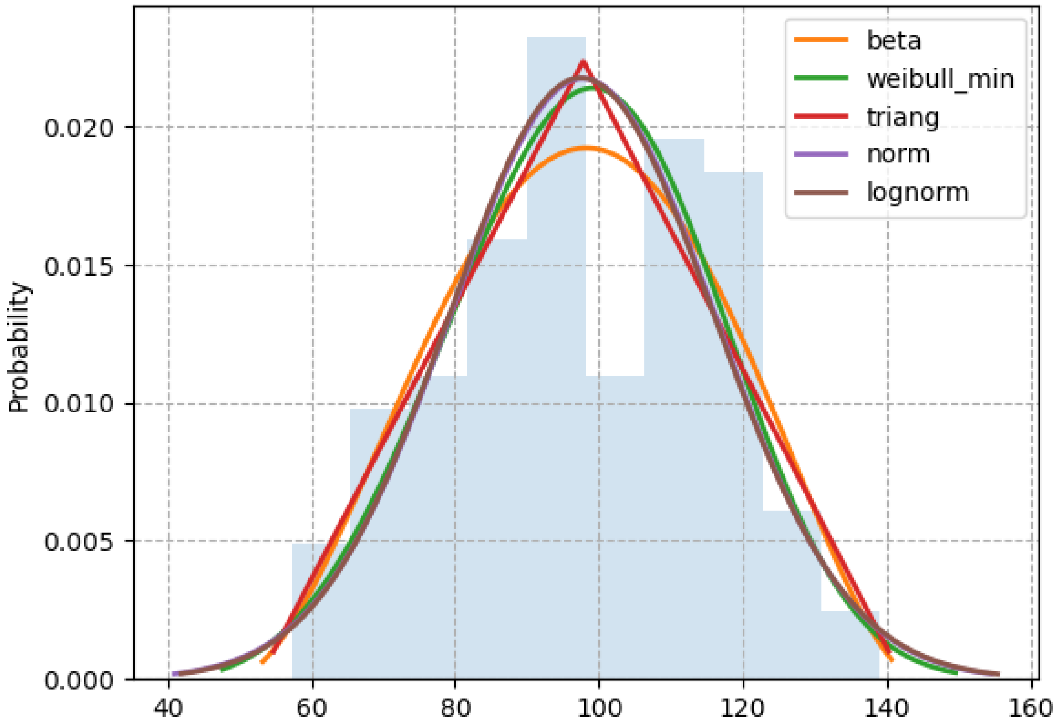

```
Data Description:
  size = 100
  minmax = (57.2, 139.09)
  mean = 97.77139999999997
  var = 337.61377604
  skewness = -0.13339778928348767
  kurtosis= -0.6831564233178393
```

[5]:
```
# Do the fits and plot the pdf and cdf of top 5 distributions
ex1.fit(Dists)
ex1.plot_pdf(5)
ex1.plot_cdf(5)
```

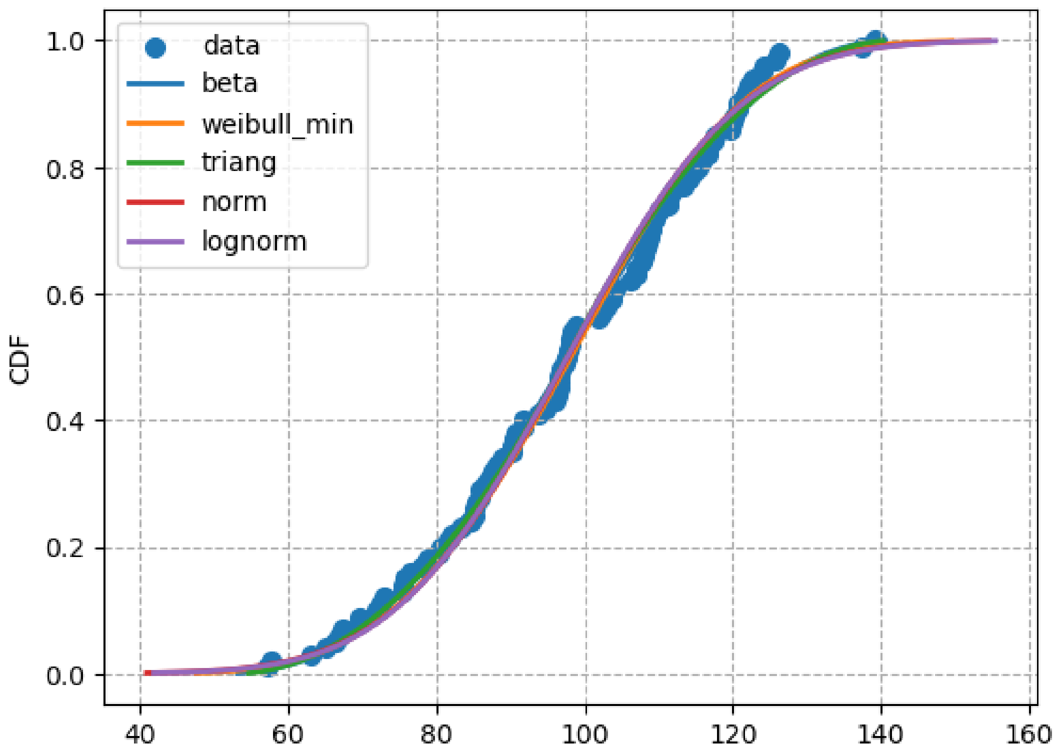
```
Number of distributions fitted = 11

Distribution 1: beta
  Parameters = (2.9694722641451423, 2.8410772308999013, 48.6010183977518,
96.0358209061009)
  KS stats = 0.0526932223744947
  p-value = 0.9302200495921993
  mean = 97.67997500732218
  var = 338.3858977910069
  sd = 18.39526835332953
```

```
Distribution 2: weibull_min
  Parameters = (3.7755301524300005, 36.82589064297723, 67.57169335277156)
  KS stats = 0.06422909438028257
  p-value = 0.779443120169669
  mean = 97.87435896417409
  var = 325.58862495960005
  sd = 18.04407451102993

Distribution 3: triang
  Parameters = (0.5036780338389795, 52.73408519584795, 89.51336326076714)
  KS stats = 0.06741904575124746
  p-value = 0.7277895637951273
  mean = 97.6005112192695
  var = 333.8661136764873
  sd = 18.272003548502482

Distribution 4: norm
  Parameters = (97.77139999999997, 18.374269401529958)
  KS stats = 0.07206510260645305
  p-value = 0.6498079161282216
  mean = 97.77139999999997
  var = 337.61377604000006
  sd = 18.374269401529958

Distribution 5: lognorm
  Parameters = (0.011160563881933781, -1547.1406362357357, 1644.
 ↪8135449430042)
  KS stats = 0.07361252519719796
  p-value = 0.6236773818384749
  mean = 97.77534939305906
  var = 337.0441281677288
  sd = 18.358761618576803

Distribution 6: logistic
  Parameters = (98.12626261718812, 10.847811913084668)
  KS stats = 0.07362021344400171
  p-value = 0.6235477725256537
  mean = 98.12626261718812
  var = 387.1353092921242
  sd = 19.675754351285345

Distribution 7: gamma
  Parameters = (535.8301275279623, -328.86401969628605, 0.
 ↪7962201909442794)
  KS stats = 0.07525756210599122
  p-value = 0.5960370046672632
  mean = 97.77474675772567
```

```
  var = 339.69840009024506
  sd = 18.430908824315882

Distribution 8: laplace
  Parameters = (97.695, 15.232199999999999)
  KS stats = 0.10838445080813608
  p-value = 0.17723194900661798
  mean = 97.695
  var = 464.03983367999996
  sd = 21.541583824779458

Distribution 9: rayleigh
  Parameters = (55.45141355800718, 32.623572984043726)
  KS stats = 0.12092263904346817
  p-value = 0.09873310581104078
  mean = 96.3389987886532
  var = 456.8004024971126
  sd = 21.37288942789703

Distribution 10: uniform
  Parameters = (57.2, 81.89)
  KS stats = 0.14134082305531814
  p-value = 0.033174856637549155  < 0.05
  mean = 98.14500000000001
  var = 558.8310083333333
  sd = 23.639606771969227

Distribution 11: expon
  Parameters = (57.2, 40.57139999999997)
  KS stats = 0.26139928766593823
  p-value = 1.6178060863787765e-06  < 0.05
  mean = 97.77139999999997
  var = 1646.0384979599974
  sd = 40.57139999999997
```



Probability
0.000
0.005
0.010
0.015
0.020
40
60
80
100
120
140
160
beta
weibull_min
triang
norm
lognorm



CDF
0.0
0.2
0.4
0.6
0.8
1.0
40
60
80
100
120
140
160
data
beta
weibull_min
triang
norm
lognorm

[6]:
```python
# Show the fitted parameters and statistics for the top 5 distributions
results = ex1.parameters(5)
```

```
The top 5 distributions are:

Distribution 1: beta
  Parameters = (2.9694722641451423, 2.8410772308999013, 48.6010183977518,
96.0358209061009)
  KS stats = 0.0526932223744947
  p-value = 0.9302200495921993
  mean = 97.67997500732218
  var = 338.3858977910069
  sd = 18.39526835332953

Distribution 2: weibull_min
  Parameters = (3.7755301524300005, 36.82589064297723, 67.57169335277156)
  KS stats = 0.06422909438028257
  p-value = 0.779443120169669
  mean = 97.87435896417409
  var = 325.58862495960005
  sd = 18.04407451102993

Distribution 3: triang
  Parameters = (0.5036780338389795, 52.73408519584795, 89.51336326076714)
  KS stats = 0.06741904575124746
  p-value = 0.7277895637951273
  mean = 97.6005112192695
  var = 333.8661136764873
  sd = 18.272003548502482

Distribution 4: norm
  Parameters = (97.77139999999997, 18.374269401529958)
  KS stats = 0.07206510260645305
  p-value = 0.6498079161282216
  mean = 97.77139999999997
  var = 337.61377604000006
  sd = 18.374269401529958

Distribution 5: lognorm
  Parameters = (0.011160563881933781, -1547.1406362357357, 1644.
↪8135449430042)
  KS stats = 0.07361252519719796
  p-value = 0.6236773818384749
  mean = 97.77534939305906
  var = 337.0441281677288
  sd = 18.358761618576803
```

5 Class DistFit_discrete

5.1 Documentation

[1]:
```python
from DecisionAnalysisPy import DistFit_discrete
print(DistFit_discrete.__doc__)
```

```
Class for fitting Discrete Distributions to Data using MLE method

   DistFit_discrete(data):
     Parameter: data = 1-D Array of data to fit.

   Attributes:
     data = Data used for fitting
     results = DataFrame of fitted results
     Distributions = Dictionary of distributions to fit

   Methods:
     data_hist(dpi=100):
         Plot relative frequencies of data
       Parameters:
         dpi = dpi to plot (default=100)

     data_describe():
         Describe the data descriptive statistics

     fit(Dists, solver='Nelder-Mead'):
       Parameters:
         Dists =  Dictionary of distributions and initial guess
               { name : tuple of parameters' initial values }
         solver = Solver for optimizing MLE
             Can be one of ('Nelder-Mead' (default) or 'Powell')
       Return:
          Nested dictionary of all fitted results

     plot_pmf(N=3, dpi=100):
         Plot the PMF of the fitted distributions and data
       Parameters:
           N = Number of top distributions to plot (default=3)
           dpi = dpi to plot (default=100)

     plot_cdf(N=3, dpi=100):
         Plot the CDF of fitted distributions and data ECDF
       Paramters:
         N = Number of top distributions to plot (default=3)
         dpi = dpi to plot (default=100)

     parameters(N=3):
```

```
    Show parameters of top fitted results
  Parameters:
    N = number of top distributions to show (default=3)
  Return:
    Nested dictioonay of fitted results
```

[ ]:

```

```

5.2 Example 2: Fitting Discrete Functions to Data

Source: 7.4.3_Fit_Discrete_Distributions_to_Data_using_DistFit_discrete.ipynb

[1]:

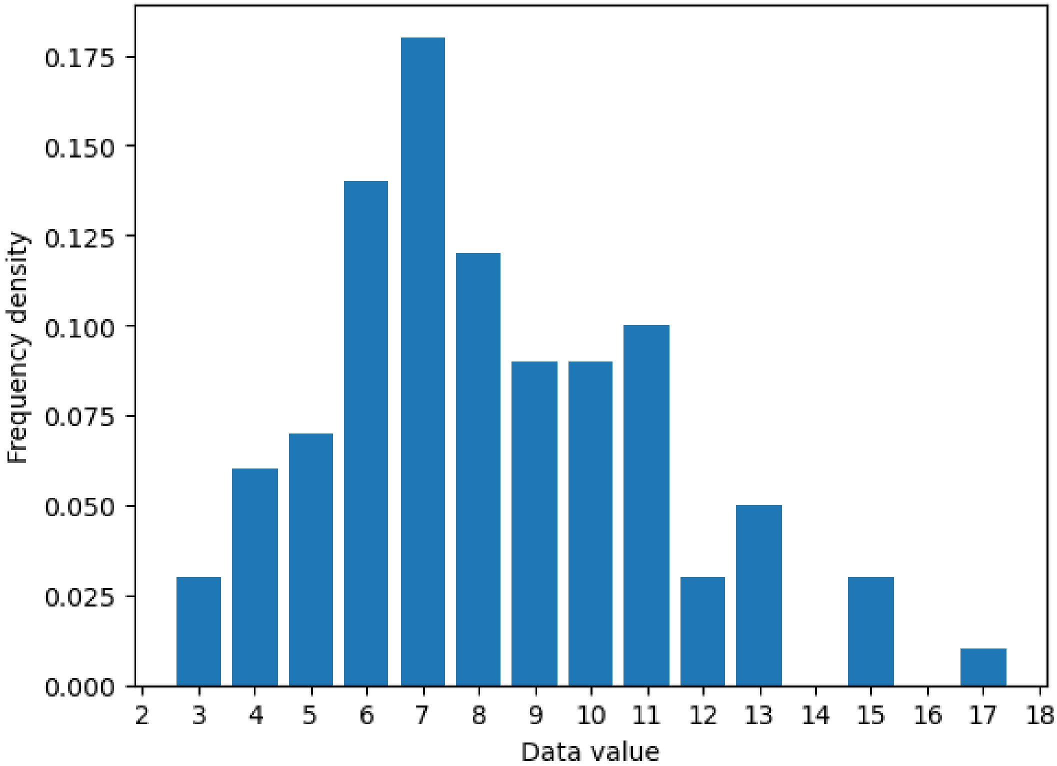
```
""" Fit discrete distributions using DistFit_discrete Class """
from DecisionAnalysisPy import DistFit_discrete
import pandas as pd
```

[2]:

```
# Read the data from Excel file
data = pd.read_excel ("7.4.3_data.xlsx", sheet_name="data2",␣
 ↪header=None)
data = data.values.flatten()
```

[3]:

```
# # Use DisFit_describe Class to fit.
ex2 = DistFit_discrete(data)

# Visualize and describe the data
ex2.data_hist(dpi=100)
# Describte the data
ex2.data_describe()
```




```
Data Description:
  size = 100
  minmax = (3, 17)
  mean = 8.18
  var = 8.4076
  skewness = 0.5715704116345816
  kurtosis= 0.0660478548040615
```

[4]:
```
#  'poisson'   # Poisson (mu)
#  'binom'     # Binomial (n, p)
#  'nbinom'    # Negative Binomial (n, p)
#  'betabinom' # Beta-Binomial (n, a, b)
#  'randint'   # randint (a, b)
#  'geom'      # geom(p)

# Distributions to fit and their parameter initial guess
Dists = { 'poisson'  : (10, ),
          'binom'    : (20, 0.5),
          'nbinom'   : (20, 0.5),
          'betabinom': (20, 10, 10),
          'randint'  : (3, 17),
          'geom'     : (0.5,)}
```

```
ex2.fit( Dists )
```

```
Fitting distribution poisson:
  MLE Optimization terminated successfully.
  Fitted Parameters = [8.17999996]

Fitting distribution binom:
  MLE Optimization terminated successfully.
  Fitted Parameters = [33.        0.24375]

Fitting distribution nbinom:
  MLE Optimization terminated successfully.
  Fitted Parameters = [308.57602061    0.97417571]

Fitting distribution betabinom:
  MLE Optimization terminated successfully.
  Fitted Parameters = [18.   10.5 10.5]

Fitting distribution randint:
```

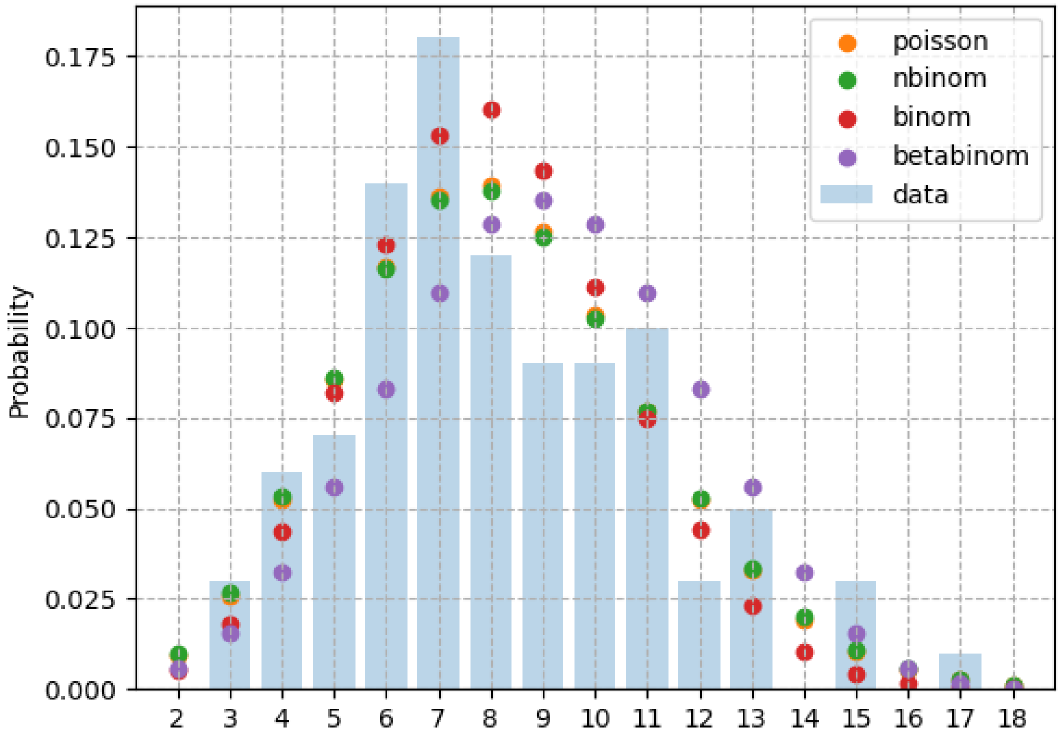
```
C:\Users\isepohkl\AppData\Local\anaconda3\lib\site-
packages\scipy\optimize\_optimize.py:863: RuntimeWarning: invalid value
encountered in subtract
  np.max(np.abs(fsim[0] - fsim[1:])) <= fatol):
```

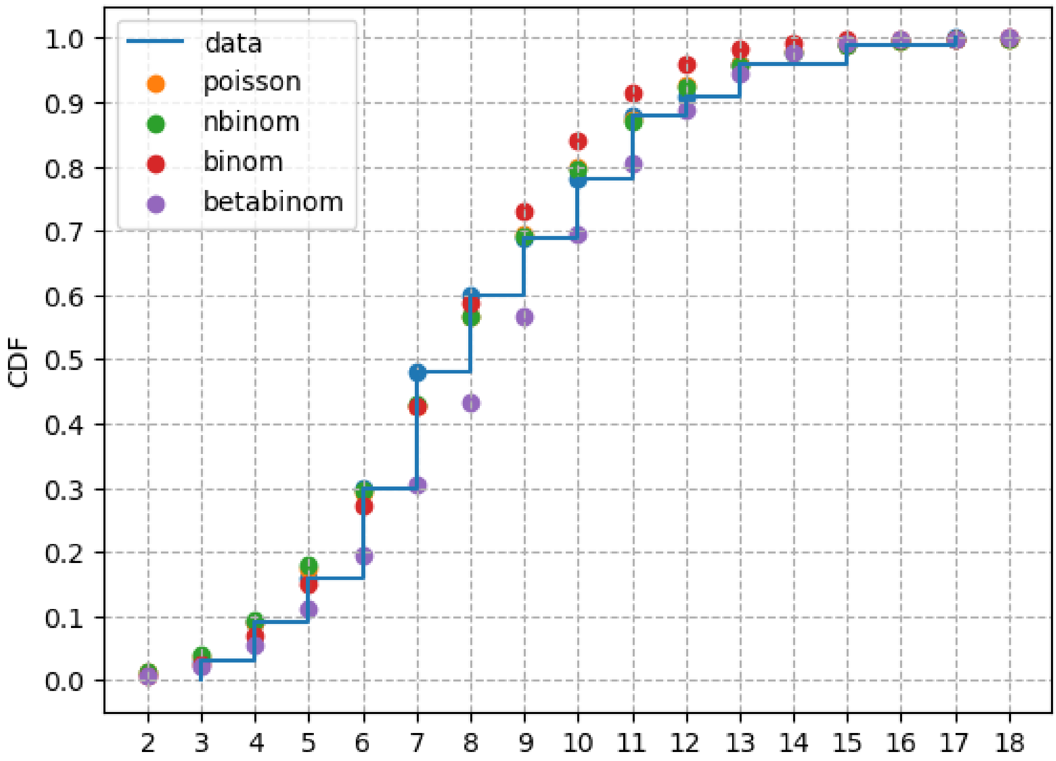
```
  MLE Maximum number of iterations has been exceeded.
  Fitted Parameters = [ 3. 17.]

Fitting distribution geom:
  MLE Optimization terminated successfully.
  Fitted Parameters = [0.12224939]

Number of distributions fitted = 5
```

[4]:
```
{'poisson': {'Params': array([8.17999996]),
  'KS_D': 0.13188736268091658,
  'KS_pv': 0.05614609245552149},
 'nbinom': {'Params': array([308.57602061,   0.97417571]),
  'KS_D': 0.13520999427877164,
  'KS_pv': 0.04686151753816992},
 'binom': {'Params': array([33.     ,  0.24375]),
  'KS_D': 0.150837808663566,
  'KS_pv': 0.018845645860325555},
 'betabinom': {'Params': array([18. , 10.5, 10.5]),
  'KS_D': 0.17607889576275854,
  'KS_pv': 0.003498486645988952},
 'geom': {'Params': array([0.12224939]),
  'KS_D': 0.3889784302638508,
```

'KS_pv': 3.9009361387286217e-14}}

[5]:
```
# Plot the results found
ex2.plot_pmf(4, dpi=100)
ex2.plot_cdf(4, dpi=100)
```





[6]:
```
# See the fitted parameters
results = ex2.parameters(4)
```

```
The top 4 distributions:

Distribution 1: poisson
  Params = [8.17999996]
  KS_stats = 0.13188736268091658
  p-value = 0.05614609245552149
  mean = 8.179999962449074
  var = 8.179999962449074
  sd = 2.8600699226503314

Distribution 2: nbinom
  Params = [308.57602061   0.97417571]
  KS_stats = 0.13520999427877164
  p-value = 0.04686151753816992  < 0.05
  mean = 8.179999455336235
  var = 8.396841940232385
  sd = 2.897730480950978

Distribution 3: binom
  Params = [33.        0.24375]
```

```
  KS_stats = 0.150837808663566
  p-value = 0.018845645860325555  < 0.05
  mean = 8.043750000000006
  var = 6.0830859375000035
  sd = 2.466391278264664

Distribution 4: betabinom
  Params = [18.  10.5 10.5]
  KS_stats = 0.17607889576275854
  p-value = 0.003498486645988952  < 0.05
  mean = 9.0
  var = 7.977272727272727275
  sd = 2.824406615073815
```

[ ]:

```

```

6 Class norm_2p

6.1 Documentation

[1]:
```
from DecisionAnalysisPy import norm_2p
```

[2]:
```
print(norm_2p.__doc__)
```

```
 Class forNormal Distribution with 2 known
        percentile values
    norm_2p(x1, q1, x2, q2):
    Parameters:
      x1, q1 = First value and its percentile
      x2, q2 = Second value and its percentile

    Attributes:
      x1, q1 = First value and its percentile
      x2, q2 = Second value and its percentile
      mu = mean of fitted distribution
      sigma = standard deviation of fitted distribution

    Methods:
      display_cdf(): Display the CDF of the distribution.
      discretize(nbr=3): Discretize the distribution
            with nbr number of branches.
```

[ ]:
```

```

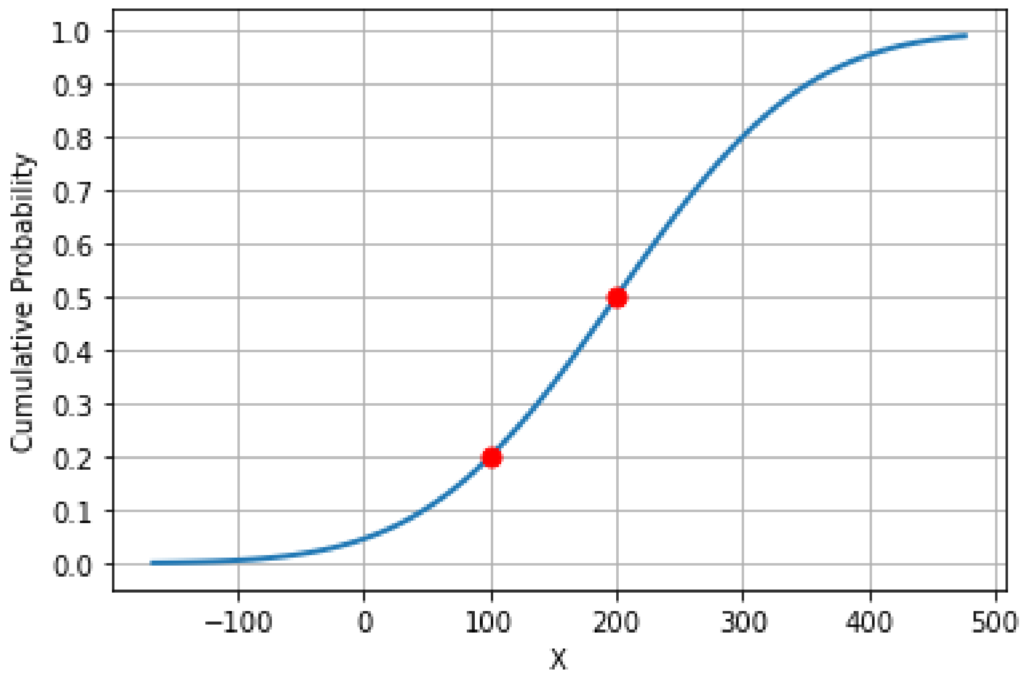
6.2 Fit Normal Distribution with 2 known percentile values

Source: 7.3.3_Fit_Normal_Dist_with_2_Percentile_Values_using_norm_2p_Class.ipynb

[1]:
```
""" Fit Normal Distribution with 2 Percentile Values using norm_2p␣
↪Class """
""" The fitted distribution is then discretized """
from DecisionAnalysisPy import norm_2p
```

[2]:
```
""" Case 1: When the mean is known """
x1, q1 = 200, 0.5
x2, q2 = 100, 0.2
```

[3]:
```
# Create a normal distribution with two known percentil values
N1 = norm_2p(x1, q1, x2, q2)
mu, sigma = N1.mu, N1.sigma
N1.display_cdf()
```



[4]:
```
print("Given:")
print(f"  {q1*100}th percentile = {x1}")
print(f"  {q2*100}th percentile = {x2}")
print("Fitted Normal distribution:")
print(f"  mean={mu}, std={sigma:.6f}")
```

```
Given:
  50.0th percentile = 200
  20.0th percentile = 100
Fitted Normal distribution:
  mean=200, std=118.818295
```

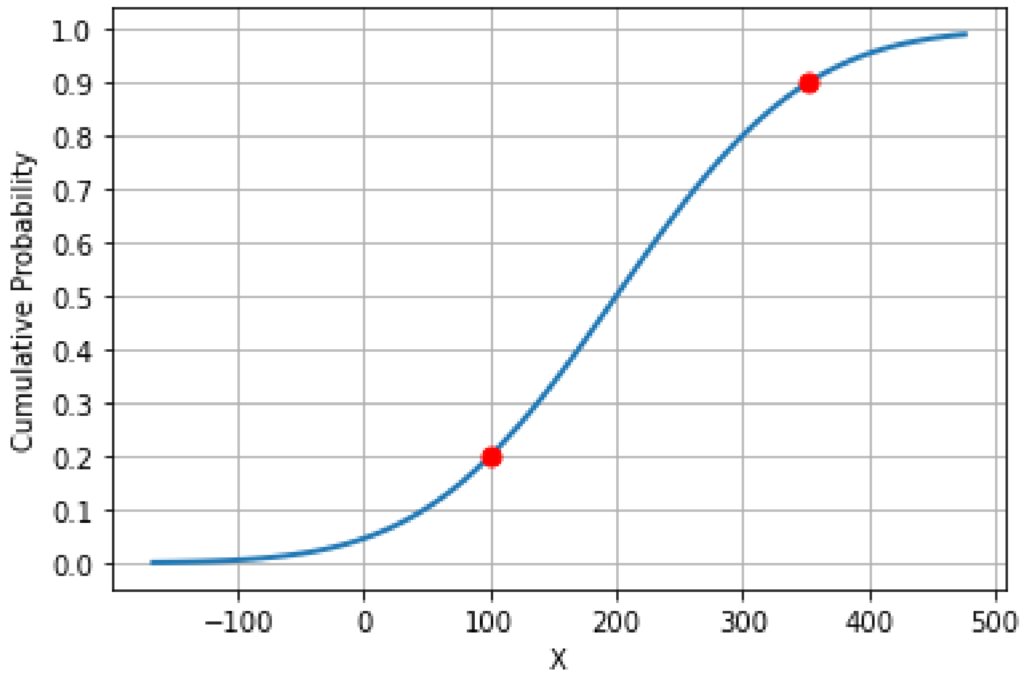
[5]:
```
# Discretize the fitted distribution
for nbr in range(1, 6):
    dx, dp = N1.discretize(nbr)
    print(f"\n{nbr}-branch discrete approximation:")
    for n in range(nbr):
        print(f"  x{n+1}={dx[n]:.2f}, p{n+1}= {dp[n]:.6f}")
```

```
1-branch discrete approximation:
  x1=200.00, p1= 1.000000

2-branch discrete approximation:
  x1=81.18, p1= 0.500000
  x2=318.82, p2= 0.500000
```

```
3-branch discrete approximation:
  x1=-5.80, p1= 0.166667
  x2=200.00, p2= 0.666667
  x3=405.80, p3= 0.166667

4-branch discrete approximation:
  x1=-77.37, p1= 0.045876
  x2=111.84, p2= 0.454124
  x3=288.16, p3= 0.454124
  x4=477.37, p4= 0.045876

5-branch discrete approximation:
  x1=-139.46, p1= 0.011257
  x2=38.93, p2= 0.222076
  x3=200.00, p3= 0.533333
  x4=361.07, p4= 0.222076
  x5=539.46, p5= 0.011257
```

[6]:
```
""" Case 2: When the mean is unknown """
x1, q1 = 100,    0.2
x2, q2 = 352.27, 0.9
```

[7]:
```
# Create a normal distribution with two known percentil values
N2 = norm_2p(x1, q1, x2, q2)
mu, sigma = N2.mu, N2.sigma
N2.display_cdf()
```



[8]:
```
print("Given:")
print(f"  {q1*100}th percentile = {x1}")
print(f"  {q2*100}th percentile = {x2}")
print("Fitted Normal distribution:")
print(f"  mean={mu}, std={sigma:.6f}")
```

```
Given:
  20.0th percentile = 100
  90.0th percentile = 352.27
Fitted Normal distribution:
  mean=199.99929759918126, std=118.817460
```

[9]:
```
# Discretize the fitted distributio
for nbr in range(1, 6):
    dx, dp = N2.discretize(nbr)
    print(f"\n{nbr}-branch discrete approximation:")
    for n in range(nbr):
        print(f"  x{n+1}={dx[n]:.2f}, p{n+1}= {dp[n]:.6f}")
```

```

1-branch discrete approximation:
  x1=200.00, p1= 1.000000

2-branch discrete approximation:
  x1=81.18, p1= 0.500000
  x2=318.82, p2= 0.500000

3-branch discrete approximation:
  x1=-5.80, p1= 0.166667
  x2=200.00, p2= 0.666667
  x3=405.80, p3= 0.166667

4-branch discrete approximation:
  x1=-77.37, p1= 0.045876
  x2=111.84, p2= 0.454124
  x3=288.16, p3= 0.454124
  x4=477.37, p4= 0.045876

5-branch discrete approximation:
  x1=-139.46, p1= 0.011257
  x2=38.93, p2= 0.222076
  x3=200.00, p3= 0.533333
  x4=361.07, p4= 0.222076
  x5=539.46, p5= 0.011257
```

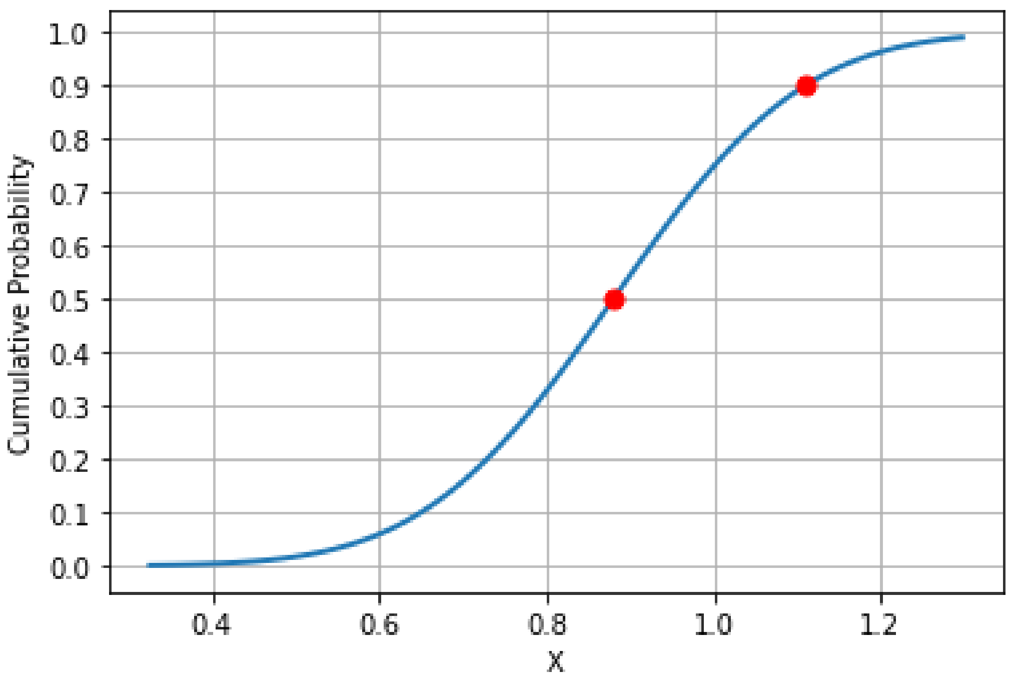
[ ]:

6.3 Exxoff Problem: Fit and Discretize Probability Distributions

Source: 8.2.2_Exxoff_Fit_and_Discretize_Distributions.ipynb

[1]:
```
""" Exxoff Problem: Fit and Discretize Probability Distributions """
from DecisionAnalysisPy import norm_2p
```

[2]:
```
# Cost of Production
x1, q1 = 0.88, 0.5
x2, q2 = 1.11, 0.9

cost = norm_2p(x1, q1, x2, q2)
mu, sigma = cost.mu, cost.sigma
cost.display_cdf()

print("Fitted Normal distribution for cost of production:")
print(f"  mean={mu}, std={sigma:.6f}")
```

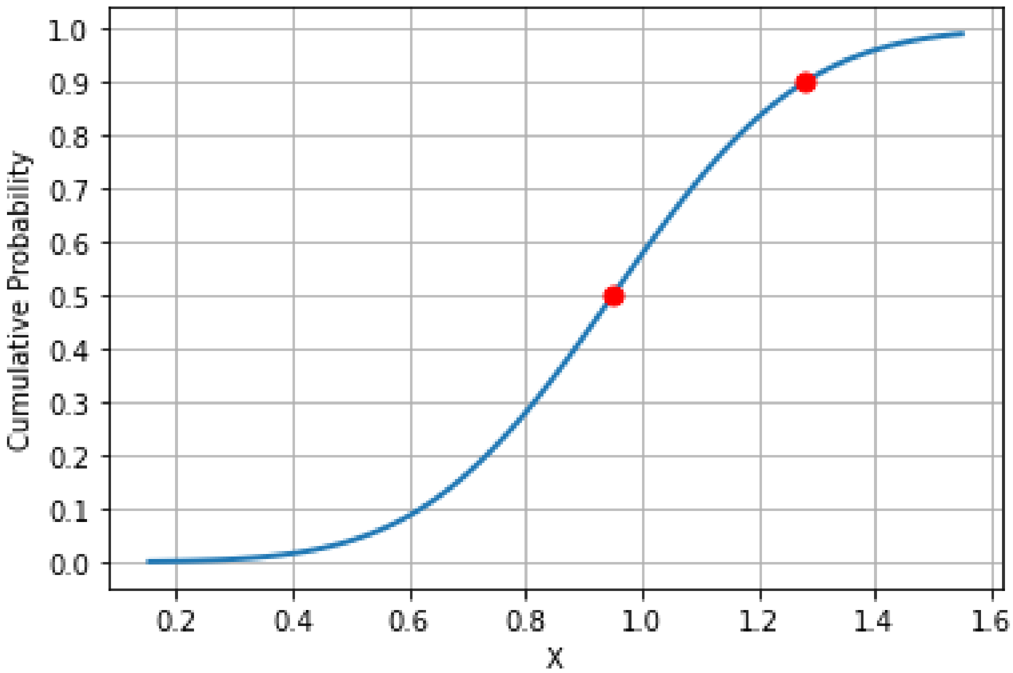



```
Fitted Normal distribution for cost of production:
  mean=0.88, std=0.179470
```

[3]:
```
# Discretize the distribution using 3 branches
nbr = 3
dx, dp = cost.discretize(nbr)
print(f"\n{nbr}-branch discrete approximation:")
for n in range(nbr):
    print(f"  x{n+1}={dx[n]:.4f}, p{n+1}= {dp[n]:.6f}")
```

```
3-branch discrete approximation:
  x1=0.5691, p1= 0.166667
  x2=0.8800, p2= 0.666667
  x3=1.1909, p3= 0.166667
```

[4]:
```
# Potency index
x1, q1 = 0.95, 0.5
x2, q2 = 1.28, 0.9

potency = norm_2p(x1, q1, x2, q2)
mu, sigma = potency.mu, potency.sigma
potency.display_cdf()
```



[5]:
```
# Discretize the distribution using 3 branches
nbr = 3
dx, dp = potency.discretize(nbr)
print(f"\n{nbr}-branch discrete approximation:")
for n in range(nbr):
    print(f"  x{n+1}={dx[n]:.4f}, p{n+1}= {dp[n]:.6f}")
```

```
3-branch discrete approximation:
  x1=0.5040, p1= 0.166667
  x2=0.9500, p2= 0.666667
  x3=1.3960, p3= 0.166667
```

7 Class OneWayRangeSensit

7.1 Documentation

[1]:
```
from DecisionAnalysisPy import OneWayRangeSensit
print(OneWayRangeSensit.__doc__)
```

```
Class for performing One-Way Range Sensitivity Analysis
   OneWayRangeSensit(v_data, f_data, obj_fns, obj_label)
   Parameters:
     v_data: uncertain variable names and their [lo, base, hi] values
     f_data: fixed variable names and thier values
     obj_funs: alternative names and their objective function␣
↪definitions
     obj_label: Output label, default = "$NPV"

  Methods:
     base_values:
         Parameters: None
         Return objective values at variable base values dictionary
     sensit_table:  Generate one-way sensitivity range table.
         Parameters: None
         Return:  Objective range values dictionary
     tornados: Plot individual tornado for each alternaive
         Parameters: annotate=True
         Return: None
     combined_tornados:  Plot all tornados together.
         Parameters:
             xlim = (lo, hi) for x-axis, default=None
             annotate = False
         Return: None
     Spiders: Plot individual tornado for each alternaive
         Parameters: None
         Return: None
```

7.2 Exxoff: One-Way Range Sensitivity Analysis

Source: 8.2.2_Exxoff_One_Way_Range_Sensitivty_Analysis.ipynb

[1]:
```
""" Exxoff Case Study One-Way Range Sensitivity Analysis"""
from DecisionAnalysisPy import OneWayRangeSensit
import numpy_financial as npf
```

[2]:
```
# Uncertain variable names and their [low, base, high] values
v_data = { 'c' : [0.65, 0.88, 1.11],
           'p' : [0.62, 0.95, 1.28],
           'f' : [0.42, 0.48, 0.54],
           'L' : [ 14,   20,   26 ] }

# Fixed parameters name and value
f_data = {'marr' : 0.1 }
```

[3]:
```
# Objective functions, one for each alternative.
# Arguments must be in the same order as above

def npv_1(c, p, f, L, marr):
    return -npf.pv(marr, 5, 0, -7-npf.pv(marr, L, 50*f*(1-p*c)))*1000
def npv_2(c, p, f, L, marr):
    return -npf.pv(marr, 5, 0, -7)*1000
def npv_3(c, p, f, L, marr):
    return 0

# The alternative names and their objective functions
obj_fns = {"Invest in R&D and Market"       : npv_1,
           "Invest in R&D and Don't market" : npv_2,
           "Don't invest in R&D"            : npv_3 }

# Label for the objective function outputs
obj_label = 'NPV($K)'
```

[4]:
```
# Perform one-way range sensitivity analysis
exoff = OneWayRangeSensit(v_data, f_data, obj_fns, obj_label)
```

[5]:
```
# Show variable and objective base values
exoff.base_values()
```

```
Variable base values:
  c = 0.88
  p = 0.95
  f = 0.48
  L = 20.00
Objective base values:
  Invest in R&D and Market =  16,460.24
```

```
Invest in R&D and Don't market =  -4,346.45
Don't invest in R&D =  0.00
```

[5]:
```
{'Invest in R&D and Market': 16460.243525945018,
 "Invest in R&D and Don't market": -4346.4492614140845,
 "Don't invest in R&D": 0}
```

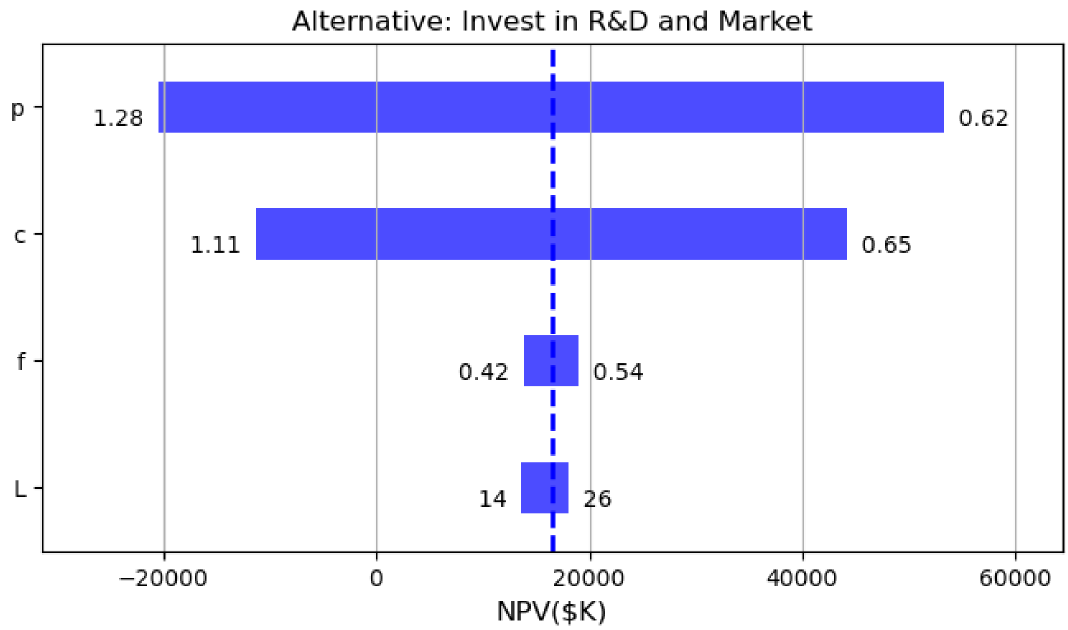
[6]:
```
# Show sensitivity range tables
exoff.sensit_table()
```

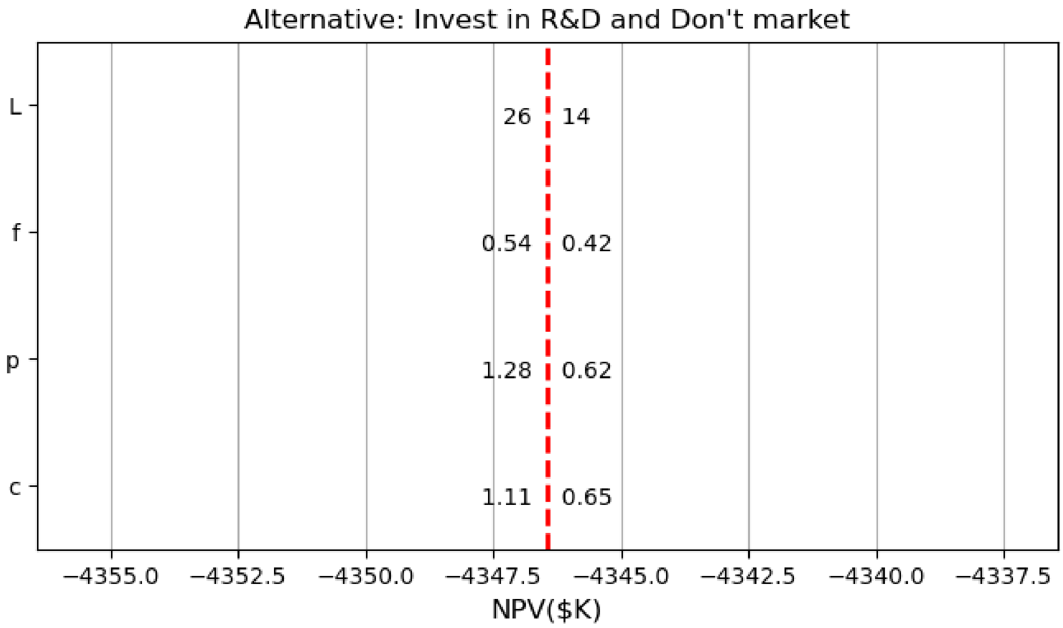
```
One-Way Range Sensitivty Tables:

Invest in R&D and Market:
  c          :       0.65       0.88       1.11 |    44,181.36   -11,260.
↪87 |
55,442.22
  p          :       0.62       0.95       1.28 |    53,303.31   -20,382.
↪83 |
73,686.14
  f          :       0.42       0.48       0.54 |    13,859.41    19,061.
↪08 |
5,201.67
  L          :      14.00      20.00      26.00 |    13,657.34    18,042.
↪41 |
4,385.07

Invest in R&D and Don't market:
  c          :       0.65       0.88       1.11 |    -4,346.45    -4,346.
↪45 |
0.00
  p          :       0.62       0.95       1.28 |    -4,346.45    -4,346.
↪45 |
0.00
  f          :       0.42       0.48       0.54 |    -4,346.45    -4,346.
↪45 |
0.00
  L          :      14.00      20.00      26.00 |    -4,346.45    -4,346.
↪45 |
0.00

Don't invest in R&D:
  c          :       0.65       0.88       1.11 |         0.00         0.
↪00 |
0.00
  p          :       0.62       0.95       1.28 |         0.00         0.
↪00 |
0.00
```

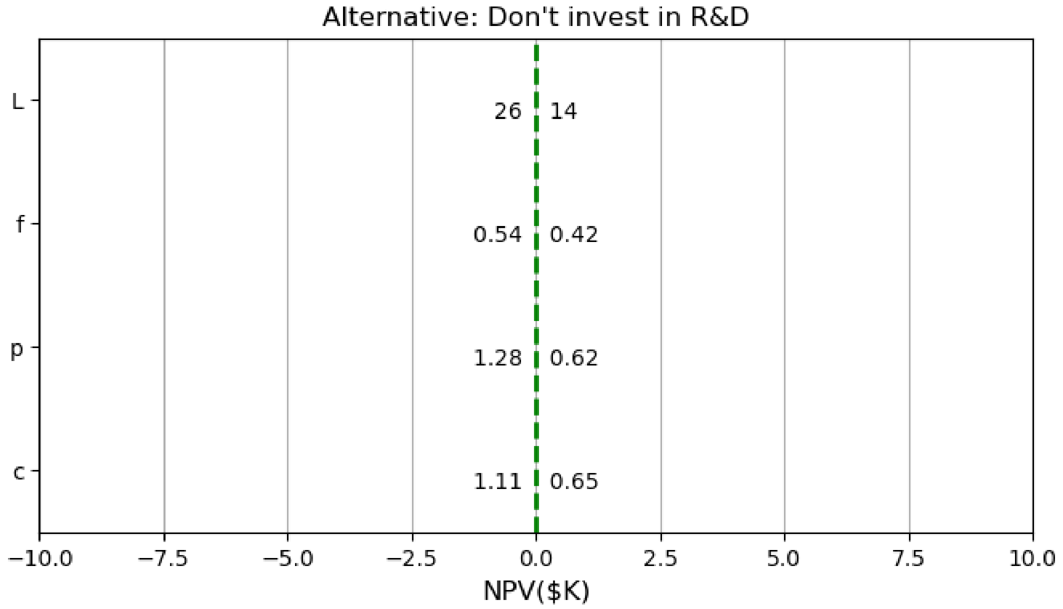
```
f          :     0.42      0.48      0.54 |       0.00      0.
↪00 |
0.00
L          :    14.00     20.00     26.00 |       0.00      0.
↪00 |
0.00
```

[6]:
```
{'Invest in R&D and Market': {'c': [44181.35556276186,
   -11260.868510871835,
   -55442.2240736337],
  'p': [53303.314168927216, -20382.827117037203, -73686.14128596442],
  'f': [13859.40692752513, 19061.080124364904, 5201.673196839774],
  'L': [13657.339495260941, 18042.40977970883, 4385.070284447887]},
 "Invest in R&D and Don't market": {'c': [-4346.4492614140845,
   -4346.4492614140845,
   0.0],
  'p': [-4346.4492614140845, -4346.4492614140845, 0.0],
  'f': [-4346.4492614140845, -4346.4492614140845, 0.0],
  'L': [-4346.4492614140845, -4346.4492614140845, 0.0]},
 "Don't invest in R&D": {'c': [0, 0, 0],
  'p': [0, 0, 0],
  'f': [0, 0, 0],
  'L': [0, 0, 0]}}
```

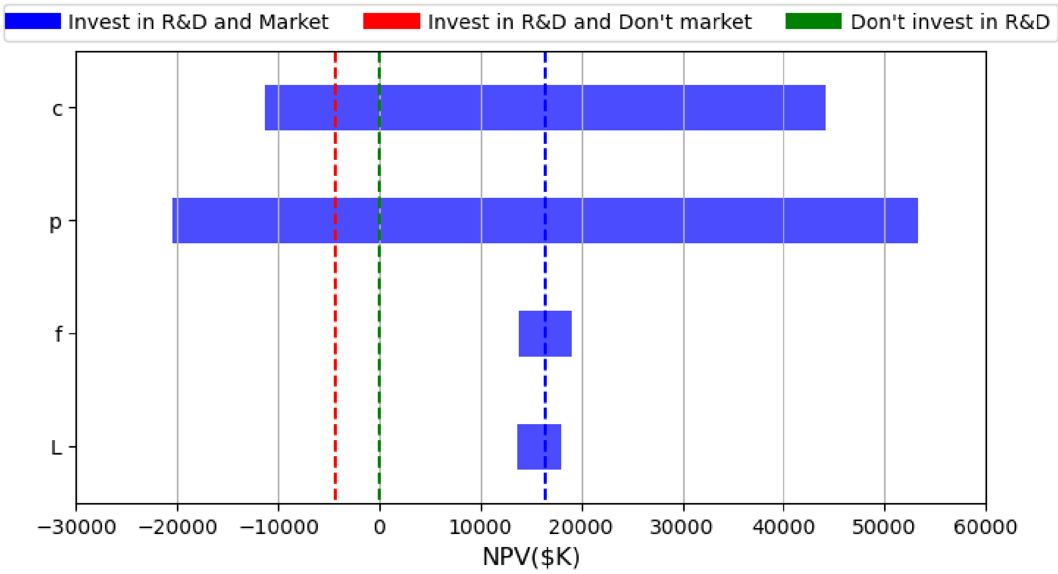
[7]:
```
# Show individual tornado diagrams
exoff.tornados()
```



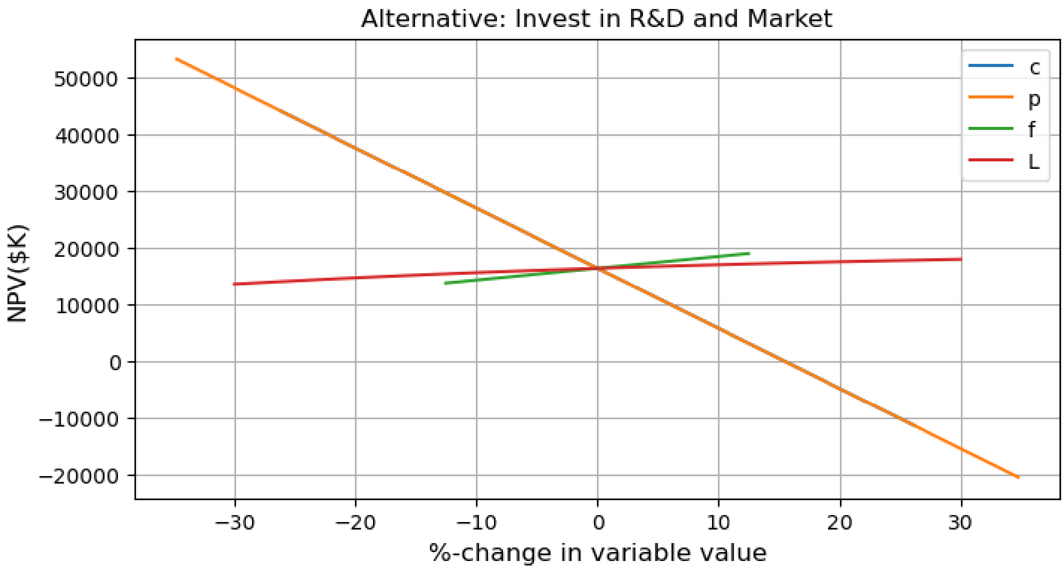


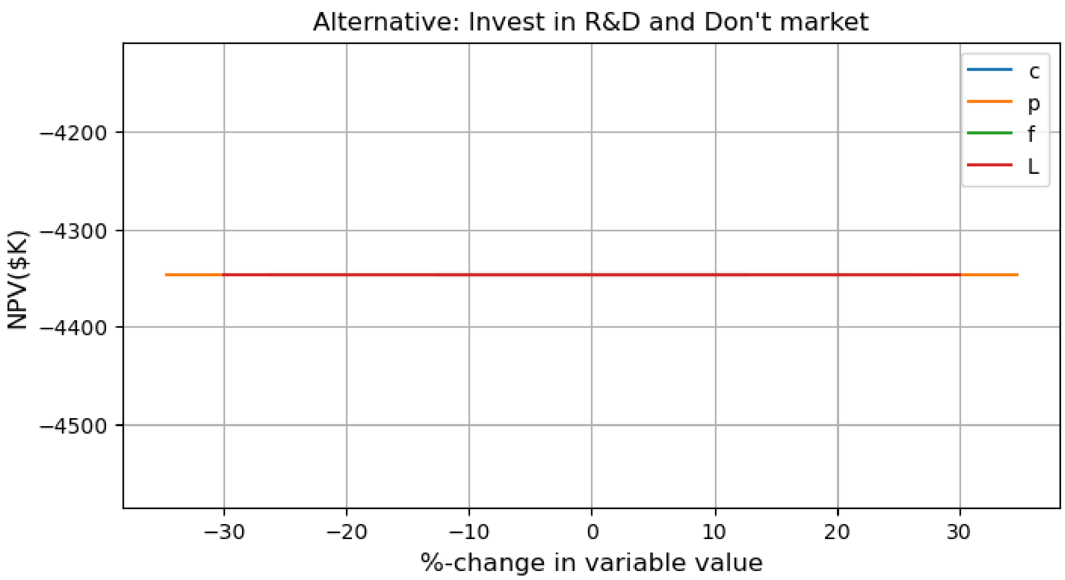


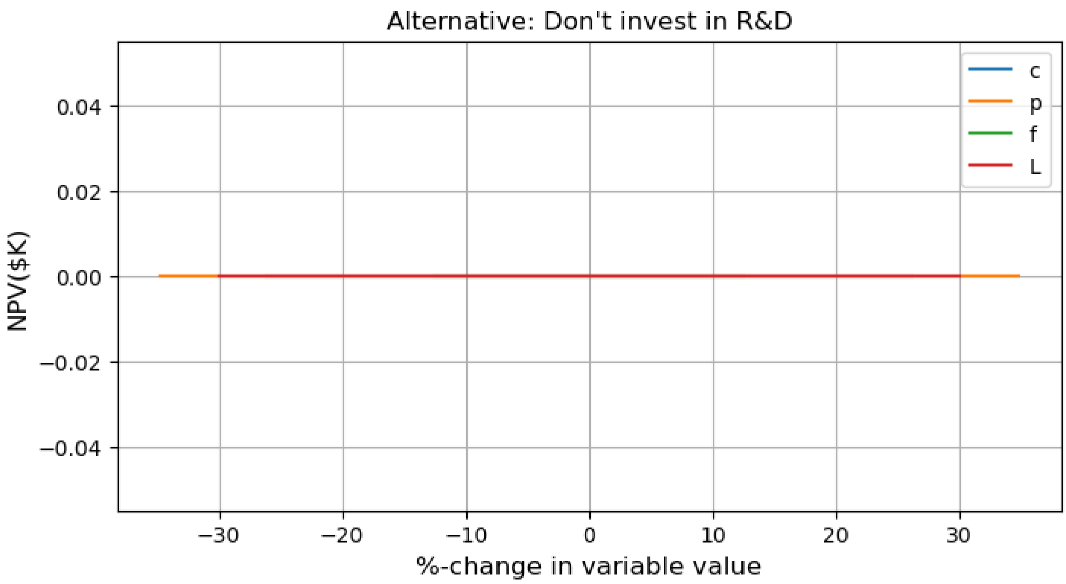
[8]:
```
# Show combined tornados
exoff.combined_tornados(xlim=(-30000,60000), annotate=False)
```



[9]:
```python
# Show individual spider diagrams
exoff.spiders()
```







[10]:
```
# Plot combined tornados for the 2 initial alterntives
def npv_invest(c, p, f, L, marr):
    return max(npv_1(c, p, f, L, marr), npv_2(c, p, f, L, marr))
    # return -npf.pv(0.1, 5, 0, -7+max(0,-npf.pv(0.1, L,␣
 ↪50*f*(1-p*c))))

init_objs = { "Invest in R&D"         : npv_invest,
              "Don't invest in R&D" : npv_3 }
```

[11]:
```
# Perform one-way range sensitivity analysis
ex_init = OneWayRangeSensit(v_data, f_data, init_objs, obj_label)
```

[12]:
```
ex_init.base_values()
# Show sensitivity range tables
```

```
Variable base values:
  c = 0.88
  p = 0.95
  f = 0.48
  L = 20.00
Objective base values:
  Invest in R&D =  16,460.24
  Don't invest in R&D =  0.00
```

[12]:
```
{'Invest in R&D': 16460.243525945018, "Don't invest in R&D": 0}
```

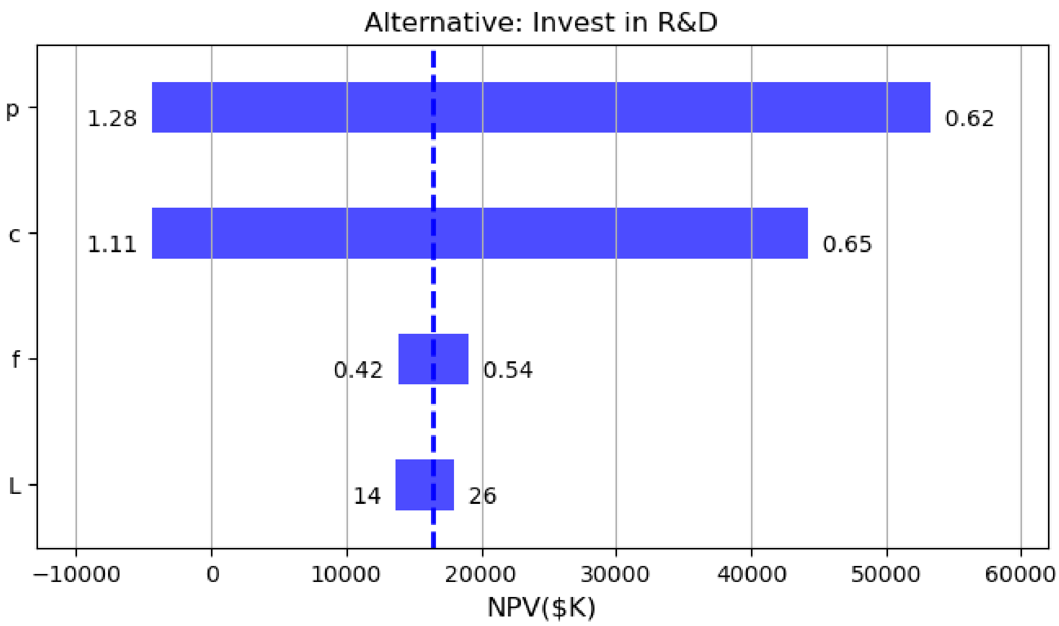
[13]:
```
ex_init.sensit_table()
# Show individual tornado diagrams
```

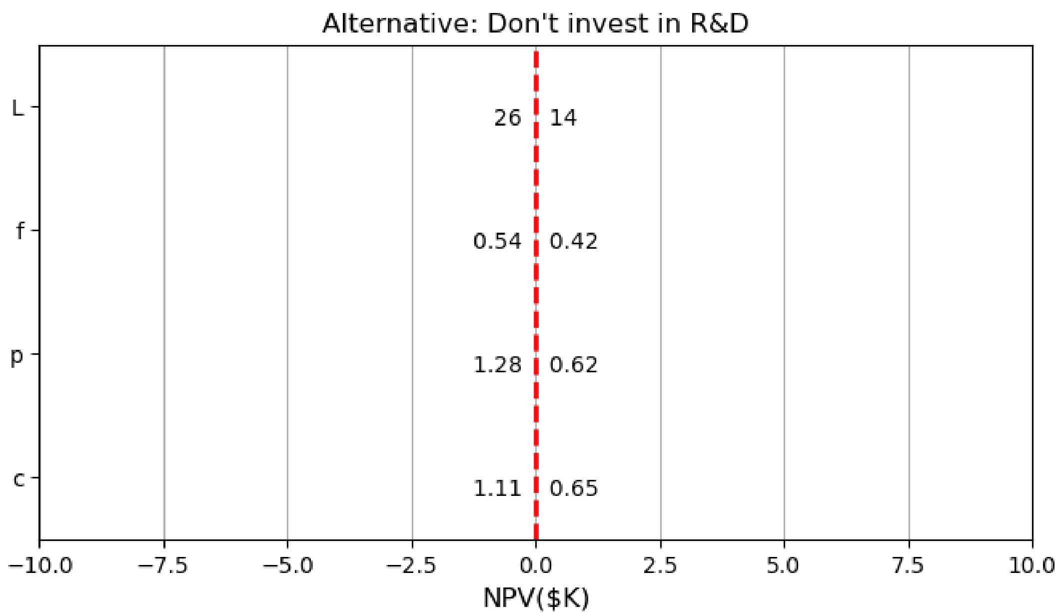
```
One-Way Range Sensitivty Tables:

Invest in R&D:
  c          :       0.65       0.88       1.11 |    44,181.36    -4,346.
 ↪45 |
48,527.80
  p          :       0.62       0.95       1.28 |    53,303.31    -4,346.
 ↪45 |
57,649.76
  f          :       0.42       0.48       0.54 |    13,859.41    19,061.
 ↪08 |
5,201.67
  L          :      14.00      20.00      26.00 |    13,657.34    18,042.
 ↪41 |
4,385.07

Don't invest in R&D:
  c          :       0.65       0.88       1.11 |         0.00         0.
 ↪00 |
0.00
  p          :       0.62       0.95       1.28 |         0.00         0.
 ↪00 |
0.00
  f          :       0.42       0.48       0.54 |         0.00         0.
 ↪00 |
0.00
```

```
  L           :       14.00       20.00       26.00 |          0.00          0.
↪00 |
0.00
```

[13]:
```
{'Invest in R&D': {'c': [44181.35556276186,
   -4346.4492614140845,
   -48527.804824175946],
  'p': [53303.314168927216, -4346.4492614140845, -57649.7634303413],
  'f': [13859.40692752513, 19061.080124364904, 5201.673196839774],
  'L': [13657.339495260941, 18042.40977970883, 4385.070284447887]},
 "Don't invest in R&D": {'c': [0, 0, 0],
  'p': [0, 0, 0],
  'f': [0, 0, 0],
  'L': [0, 0, 0]}}
```

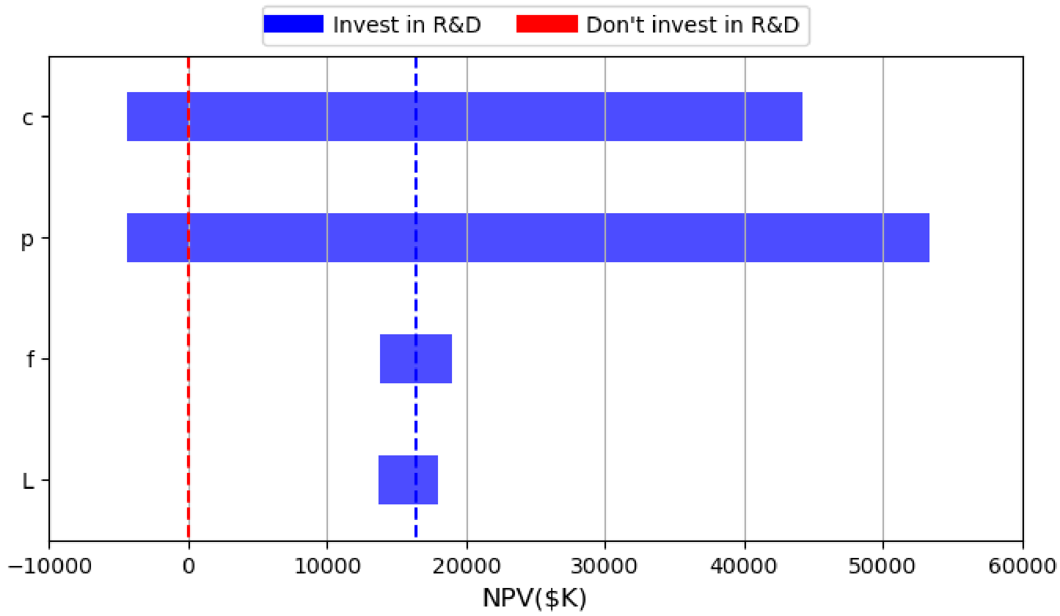
[14]:
```
ex_init.tornados()
# Show combined tornados
```





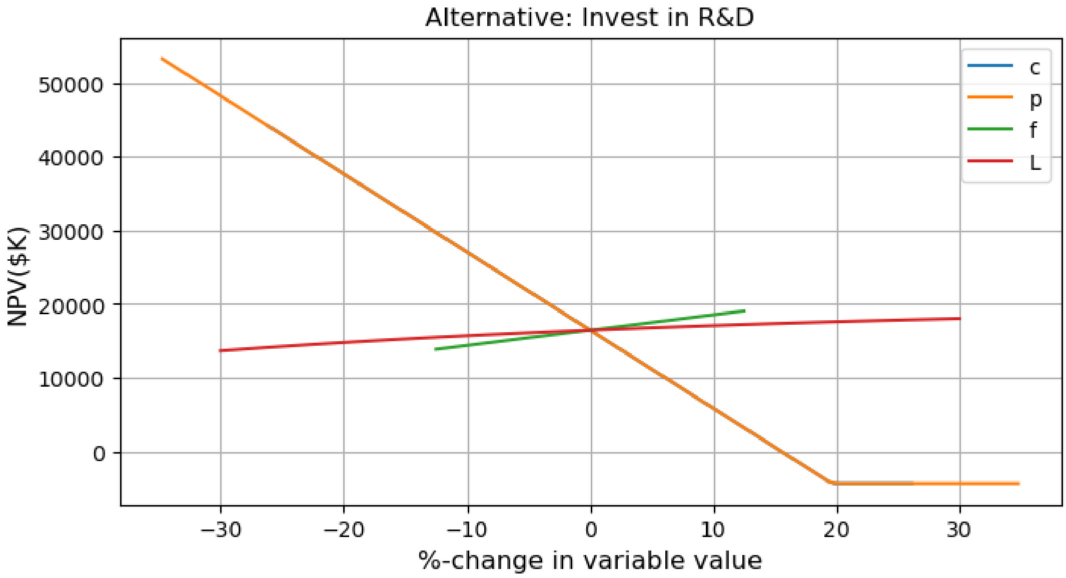
[15]:

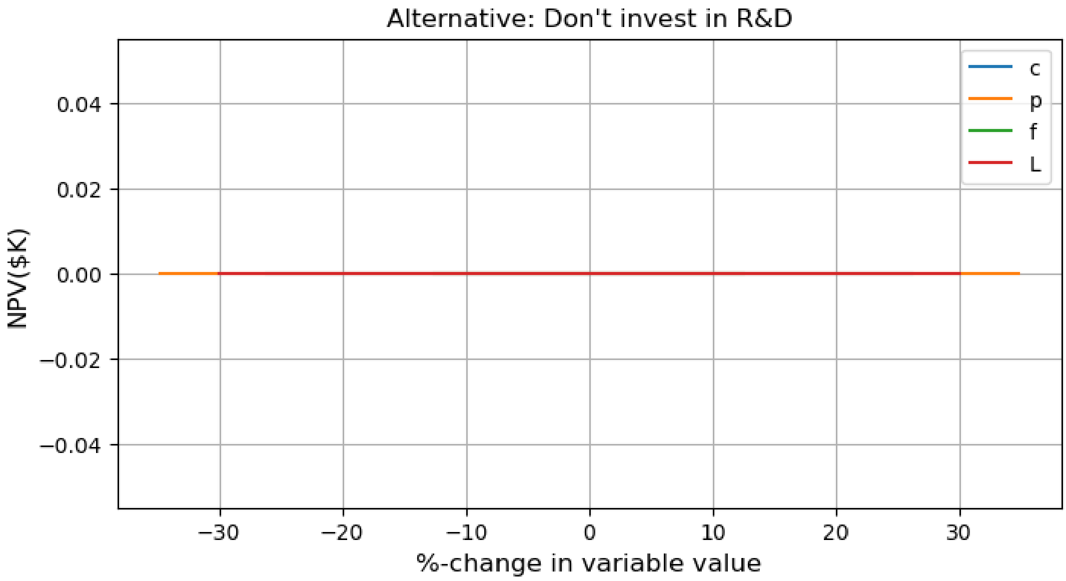
```
# Show combined tornados
ex_init.combined_tornados(xlim=(-10000,60000), annotate=False)
```



[16]:

```
# Show individual spider diagrams
ex_init.spiders()
```


Alternative: Invest in R&D
c
p
f
L
NPV($K)
50000
40000
30000
20000
10000
0
−30
−20
−10
0
10
20
30
%-change in variable value


Alternative: Don't invest in R&D
c
p
f
L
NPV($K)
0.04
0.02
0.00
−0.02
−0.04
−30
−20
−10
0
10
20
30
%-change in variable value

[ ]:

7.3 LIM: One-Way Range Sensitivity Analysis

Source: 8.4.2_LIM_One_Way_Range_Sensitivty_Analysis.ipynb

[1]:
```
""" LIM Case Study: One-Way Range Sensitivity Analysis """
from DecisionAnalysisPy import OneWayRangeSensit
import numpy_financial as npf
""" LIM Marketing and Service Support Case Study (2025 01 26) """
```

[1]:
```
' LIM Marketing and Service Support Case Study (2025 01 26) '
```

[2]:
```
# Uncertain variable names and their [low, base, high] values
v_data = {   'nsold'   : [  25,    30,    50],
             'c_alpha' : [   9,    10,    11],
             'alpha'   : [0.13, 0.15, 0.17],
             'c_beta'  : [ 7.8,   8.0,   8.1],
             'beta'    : [0.68, 0.70, 0.72],
             'trg_c'   : [   4,     8,    10]}

# Fixed parameters name and value
f_data = {'marr' : 0.03 }
```

[3]:
```
# Objective functions, one for each alternative.
# Arguments must be in the same order as above
def npv_1(nsold, c_alpha, alpha, c_beta, beta, trg_c, marr):
    return nsold*(30-(-npf.pv(marr, 5, alpha*c_alpha +␣
 ↪beta*c_beta)))*1000
def npv_2(nsold, c_alpha, alpha, c_beta, beta, trg_c, marr):
    return nsold*(30-(-npf.pv(marr, 5, alpha*c_alpha)) -10 - trg_c)*1000
def npv_3(nsold, c_alpha, alpha, c_beta, beta, trg_c, marr):
    return (nsold*30 - max(750, 23*nsold))*1000
def npv_4(nsold, c_alpha, alpha, c_beta, beta, trg_c, marr):
    return -25.0*1000

# The alternative names and their objective functions
obj_fns = { "Present Arrangement" : npv_1,
            "Train user"          : npv_2,
            "Contract IPX"        : npv_3,
            "Withdraw from market": npv_4 }

# Label for the objective function outputs
obj_label = "NPV($)"
```

[4]:
```
# Perform one-way range sensitivity analysis
Lim = OneWayRangeSensit(v_data, f_data, obj_fns, obj_label)
```

[5]:
```
# Show variable and objective base values
Lim.base_values()
```

```
Variable base values:
  nsold = 30.00
  c_alpha = 10.00
  alpha = 0.15
  c_beta = 8.00
  beta = 0.70
  trg_c = 8.00
Objective base values:
  Present Arrangement =  -75,477.63
  Train user =  153,913.18
  Contract IPX =  150,000.00
  Withdraw from market =  -25,000.00
```

[5]:
```
{'Present Arrangement': -75477.63087243616,
 'Train user': 153913.17657624587,
 'Contract IPX': 150000,
 'Withdraw from market': -25000.0}
```

[6]:
```
# Show sensitivity range tables
Lim.sensit_table()
```

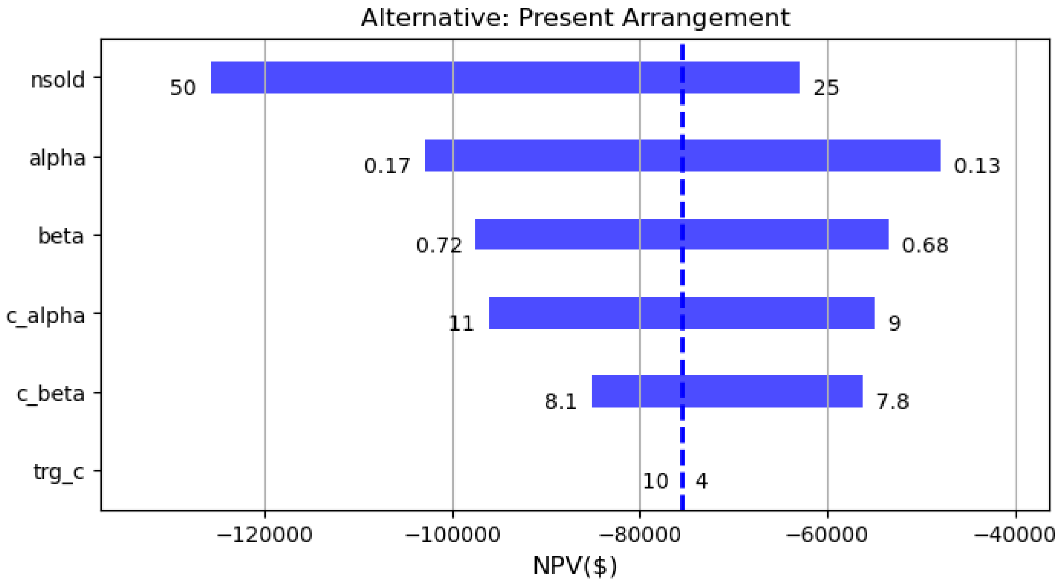
```
One-Way Range Sensitivty Tables:

Present Arrangement:
  nsold      :      25.00      30.00      50.00 |   -62,898.03  -125,796.
 ↪05 |
62,898.03
  c_alpha    :       9.00      10.00      11.00 |   -54,868.95   -96,086.
 ↪31 |
41,217.36
  alpha      :       0.13       0.15       0.17 |   -47,999.39  -102,955.
 ↪87 |
54,956.49
  c_beta     :       7.80       8.00       8.10 |   -56,242.86   -85,095.
 ↪02 |
28,852.16
  beta       :       0.68       0.70       0.72 |   -53,495.04   -97,460.
 ↪23 |
43,965.19
  trg_c      :       4.00       8.00      10.00 |   -75,477.63   -75,477.
 ↪63 |
0.00

Train user:
  nsold      :      25.00      30.00      50.00 |   128,260.98   256,521.
 ↪96 |
```

```
128,260.98
  c_alpha    :        9.00        10.00        11.00 |   174,521.86    133,304.
 ↪49 |
41,217.36
  alpha      :        0.13         0.15         0.17 |   181,391.42    126,434.
 ↪93 |
54,956.49
  c_beta     :        7.80         8.00         8.10 |   153,913.18    153,913.
 ↪18 |
0.00
  beta       :        0.68         0.70         0.72 |   153,913.18    153,913.
 ↪18 |
0.00
  trg_c      :        4.00         8.00        10.00 |   273,913.18     93,913.
 ↪18 |
180,000.00

Contract IPX:
  nsold      :       25.00        30.00        50.00 |         0.00    350,000.
 ↪00 |
350,000.00
  c_alpha    :        9.00        10.00        11.00 |   150,000.00    150,000.
 ↪00 |
0.00
  alpha      :        0.13         0.15         0.17 |   150,000.00    150,000.
 ↪00 |
0.00
  c_beta     :        7.80         8.00         8.10 |   150,000.00    150,000.
 ↪00 |
0.00
  beta       :        0.68         0.70         0.72 |   150,000.00    150,000.
 ↪00 |
0.00
  trg_c      :        4.00         8.00        10.00 |   150,000.00    150,000.
 ↪00 |
0.00

Withdraw from market:
  nsold      :       25.00        30.00        50.00 |   -25,000.00    -25,000.
 ↪00 |
0.00
  c_alpha    :        9.00        10.00        11.00 |   -25,000.00    -25,000.
 ↪00 |
0.00
  alpha      :        0.13         0.15         0.17 |   -25,000.00    -25,000.
 ↪00 |
0.00
```

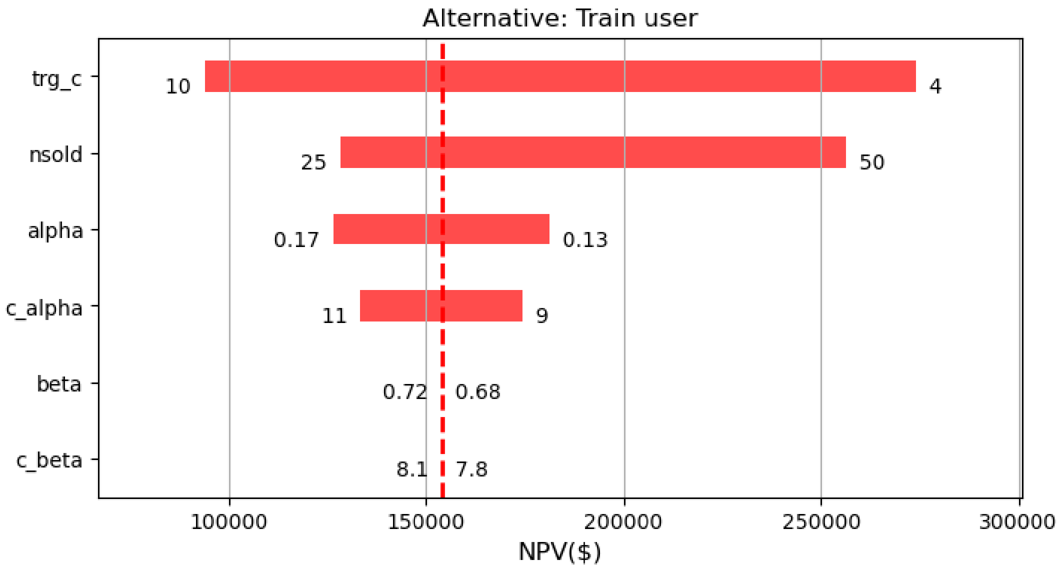
```
 c_beta    :       7.80       8.00       8.10 |   -25,000.00   -25,000.
↪00 |
0.00
 beta      :       0.68       0.70       0.72 |   -25,000.00   -25,000.
↪00 |
0.00
 trg_c     :       4.00       8.00      10.00 |   -25,000.00   -25,000.
↪00 |
0.00
```

[6]:
```
{'Present Arrangement': {'nsold': [-62898.02572703014,
   -125796.05145406027,
   -62898.02572703014],
  'c_alpha': [-54868.94853006063, -96086.31321481148, -41217.
↪36468475085],
  'alpha': [-47999.38774926875, -102955.87399560347, -54956.
↪486246334716],
  'c_beta': [-56242.8606862192, -85095.01596554443, -28852.
↪155279325227],
  'beta': [-53495.03637390239, -97460.22537096984, -43965.18899706745],
  'trg_c': [-75477.63087243616, -75477.63087243616, 0.0]},
 'Train user': {'nsold': [128260.98048020489,
   256521.96096040978,
   128260.98048020489],
  'c_alpha': [174521.8589186213, 133304.49423387053, -41217.
↪36468475076],
  'alpha': [181391.41969941306, 126434.93345307867, -54956.48624633439],
  'c_beta': [153913.17657624587, 153913.17657624587, 0.0],
  'beta': [153913.17657624587, 153913.17657624587, 0.0],
  'trg_c': [273913.17657624587, 93913.17657624587, -180000.0]},
 'Contract IPX': {'nsold': [0, 350000, 350000],
  'c_alpha': [150000, 150000, 0],
  'alpha': [150000, 150000, 0],
  'c_beta': [150000, 150000, 0],
  'beta': [150000, 150000, 0],
  'trg_c': [150000, 150000, 0]},
 'Withdraw from market': {'nsold': [-25000.0, -25000.0, 0.0],
  'c_alpha': [-25000.0, -25000.0, 0.0],
  'alpha': [-25000.0, -25000.0, 0.0],
  'c_beta': [-25000.0, -25000.0, 0.0],
  'beta': [-25000.0, -25000.0, 0.0],
  'trg_c': [-25000.0, -25000.0, 0.0]}}
```

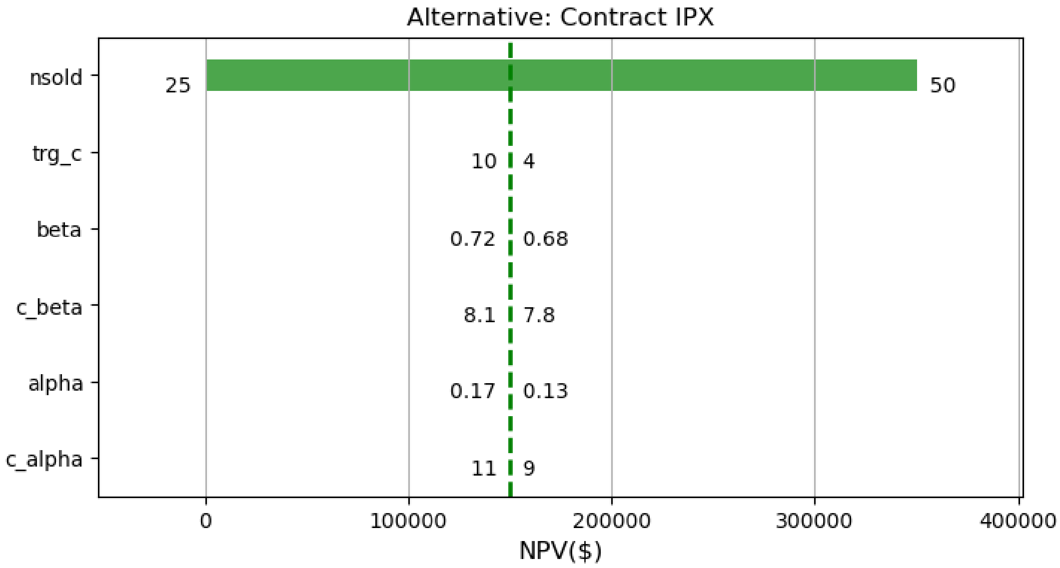
[7]:
```python
# Show individual tornado diagrams
Lim.tornados(annotate=True)
```

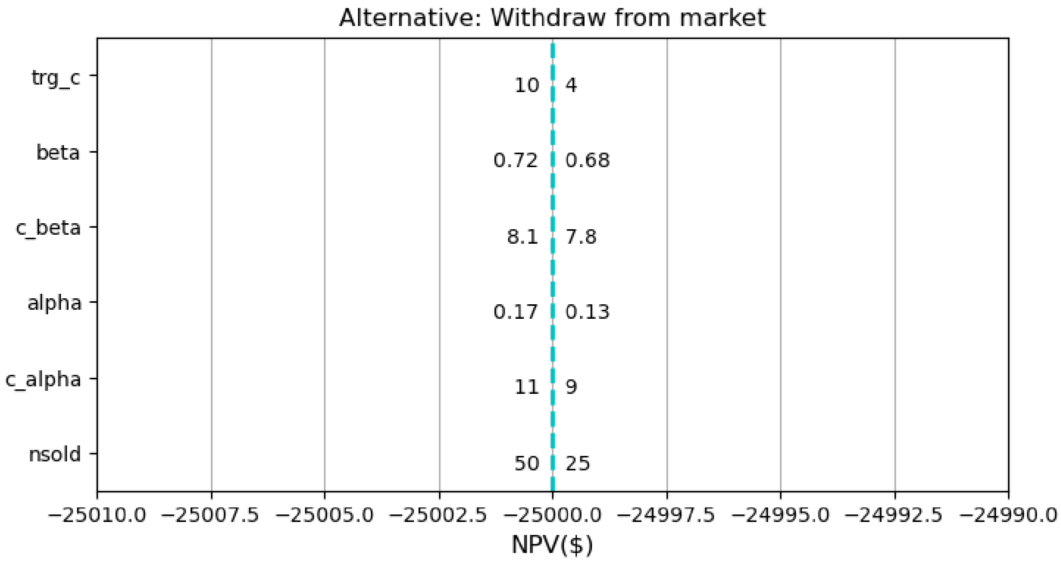


Alternative: Present Arrangement
nsold
alpha
beta
c_alpha
c_beta
trg_c
50
25
0.17
0.13
0.72
0.68
11
9
8.1
7.8
10
4
−120000
−100000
−80000
−60000
−40000
NPV($)

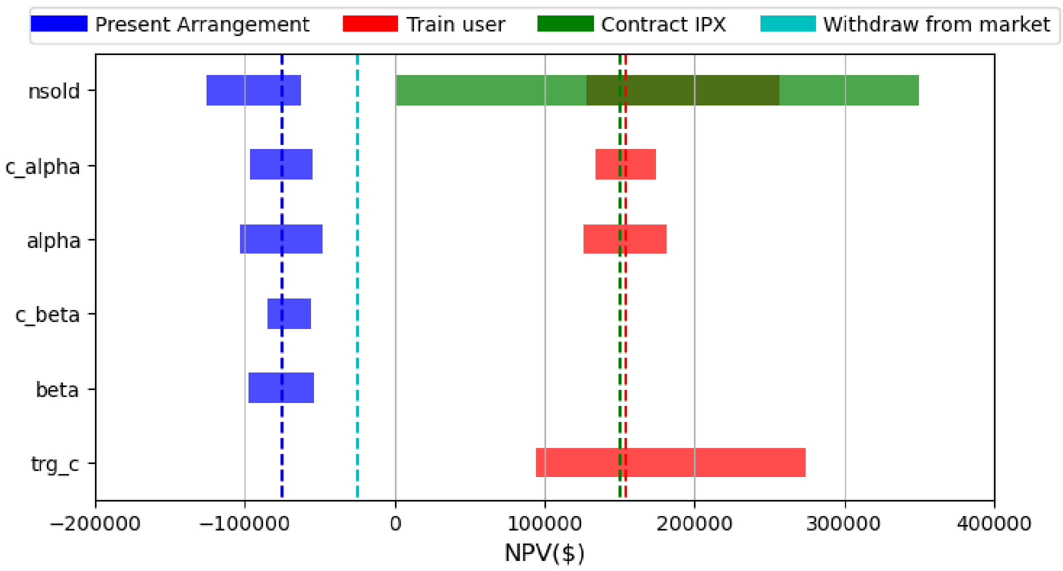


Alternative: Train user
trg_c
nsold
alpha
c_alpha
beta
c_beta
10
4
25
50
0.17
0.13
11
9
0.72
0.68
8.1
7.8
100000
150000
200000
250000
300000
NPV($)

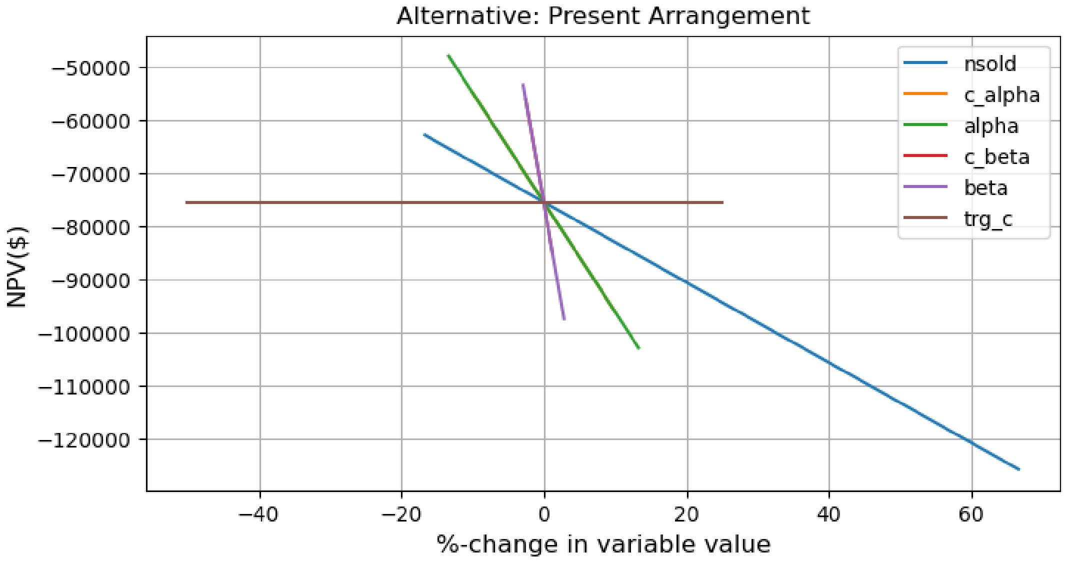


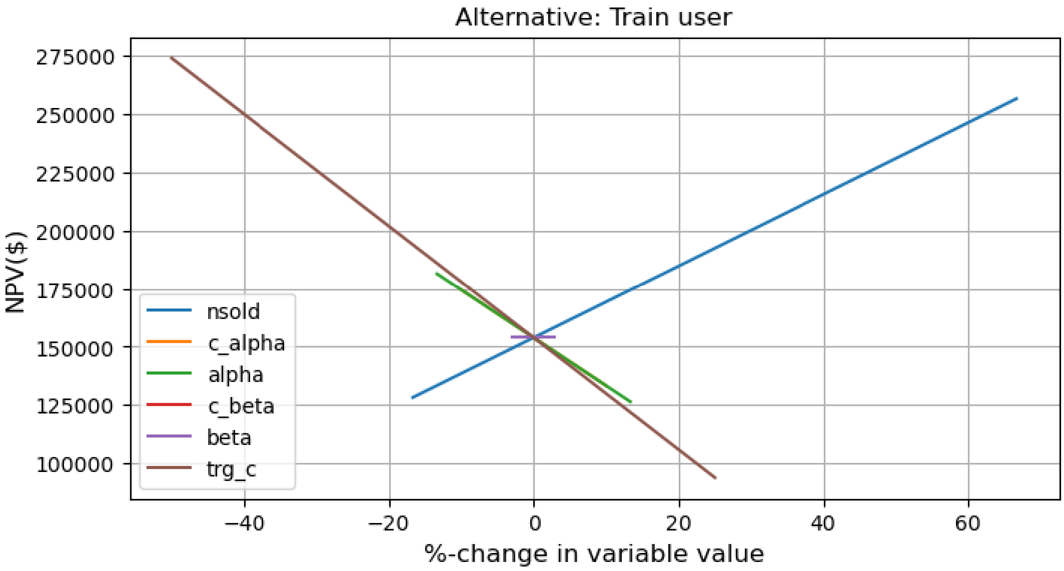


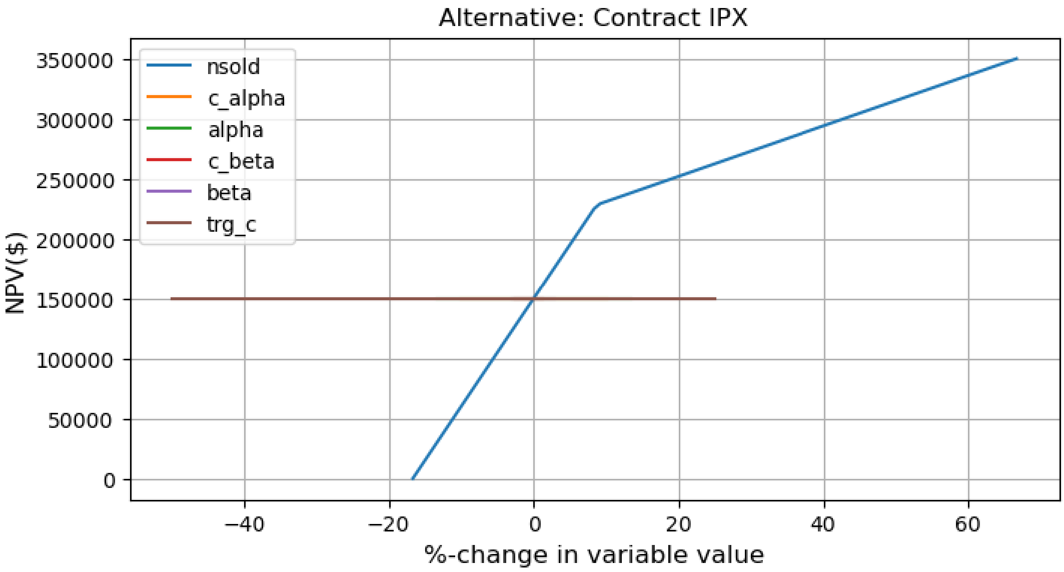
[8]:
```
# Show combined tornados
Lim.combined_tornados((-200000, 400000), annotate=False)
```

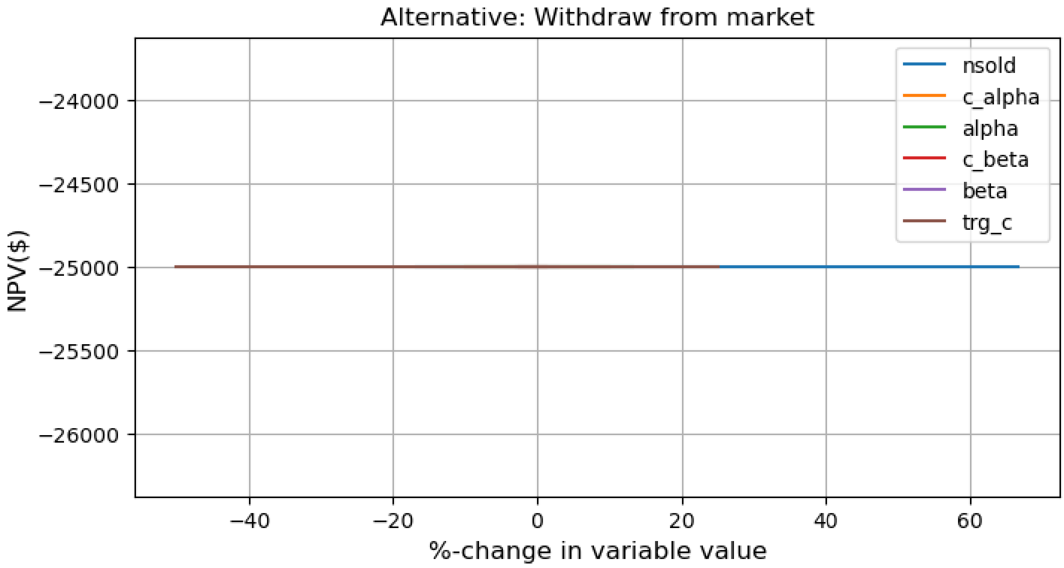


[9]:
```
# Show individual spider diagrams
Lim.spiders()
```




Alternative: Train user
275000
250000
225000
200000
175000
150000
125000
100000
NPV($)
nsold
c_alpha
alpha
c_beta
beta
trg_c
−40
−20
0
20
40
60
%-change in variable value


Alternative: Contract IPX
350000
300000
250000
200000
150000
100000
50000
0
NPV($)
nsold
c_alpha
alpha
c_beta
beta
trg_c
−40
−20
0
20
40
60
%-change in variable value


Alternative: Withdraw from market
−24000
−24500
−25000
−25500
−26000
NPV($)
−40
−20
0
20
40
60
%-change in variable value
nsold
c_alpha
alpha
c_beta
beta
trg_c

```
[ ]:
```

8 Class AHPmatrix

8.1 Documentation

[1]:
```
from DecisionAnalysisPy import AHPmatrix
```

[2]:
```
print(AHPmatrix.__doc__)
```

```
Class for AHP pairwise comparison matrices
   AHPmatrix(A, upper_triangle=False, description=''):
   Parameters:
     A = a valid AHP pairwise comparison matrix
          upper_triangle: True if A is upper triangle,
                         False (default) if otherwise.
     description = doc string for A

   Attributes:
     A = the pairwise comparison matrix
     w = the priority weight after A is solved
     size = the size of the matrix A
     lambda_max = the dominance eigen value of A
     CI = the inconsistency index of A
     CR = the inconsistency ratio of A

   Methods:
     solve(method):
       method = "Algebra"(default),"Power", "RGM",
                   "ColsNorm", "GenEigen"
     pprint(): Pretty print matrix A
```

[ ]:
```

```

8.2 Compute AHP matrices using different methods

Source: 9.2.3_Compute_AHP_matrix_using_AHPmatrix_Class.ipynb

[1]:
```
""" Compute AHP Matrix using different methods with AHPmatrix Class """
from DecisionAnalysisPy import AHPmatrix
import numpy as np
```

[2]:
```
A3 = AHPmatrix([[1, 1/3, 1/2],
                [ 3,  1,  3 ],
                [ 2, 1/3, 1 ]])
```

[3]:
```
# Pretty print the matrix
A3.pprint()
```

```
[[ 1   1/3  1/2 ]
 [ 3    1    3  ]
 [ 2   1/3   1  ]]
```

[4]:
```python
# Evaluate the matarix using 5 diferent methods
for method in ["Algebra", "Power", "RGM", "ColsNorm", "GenEigen"]:
    w, lam, ci, cr = A3.solve(method)
    print(f"method = {method}")
    np.set_printoptions(precision=6, suppress=True)
    print(f"  w={w}")
    print(f"  lambda_max={lam:.6f},  CI={ci:6f},  CR={cr:6f}")
```

```
method = Algebra
  w=[0.157056 0.593634 0.249311]
  lambda_max=3.053622,  CI=0.026811,  CR=0.046225
method = Power
  w=[0.157056 0.593634 0.249311]
  lambda_max=3.053622,  CI=0.026811,  CR=0.046225
method = RGM
  w=[0.157056 0.593634 0.249311]
  lambda_max=3.053622,  CI=0.026811,  CR=0.046225
method = ColsNorm
  w=[0.159259 0.588889 0.251852]
  lambda_max=3.053904,  CI=0.026952,  CR=0.046469
method = GenEigen
  w=[0.157056 0.593634 0.249311]
  lambda_max=3.053622,  CI=0.026811,  CR=0.046225
```

[5]:
```python
# You can also get individual attrbutes after evaluating the matrix
A3.solve(method='Power')
print(f"\nsize = {A3.size}")
print(f"w = {A3.w}")
print(f"lambda_max={A3.lambda_max:.6f}, CI={A3.CI:.6f}, CR={A3.CR:.6f}")
```

```

size = 3
w = [0.157056 0.593634 0.249311]
lambda_max=3.053622, CI=0.026811, CR=0.046225
```

[6]:
```python
# We can also enter just the upper triangle
T3 = AHPmatrix([1/3, 1/2, 3], upper_triangle=True)
T3.pprint()
w, lam, ci, cr = T3.solve("Algebra")
print(f"  w={w}")
print(f"  lambda_max={lam:.6f},  CI={ci:6f},  CR={cr:6f}")
```

```
[[ 1   1/3  1/2 ]
 [ 3    1    3  ]
 [ 2   1/3   1  ]]
  w=[0.157056 0.593634 0.249311]
```

```
lambda_max=3.053622,  CI=0.026811,  CR=0.046225
```

[ ]:

```

```

9 Class AHP3Lmodel

9.1 Documentation

[1]:
```
from DecisionAnalysisPy import AHP3Lmodel
```

[2]:
```
print(AHP3Lmodel.__doc__)
```

```
Class for 3-Level AHP Models
   AHP3model(Goal, MC, MC_matrix, alt_names, alt_matrices)
   Parameters:
     Goal = Goal.
     MC = list of main criteria.
     MC_matrix = upper triangle of criteria pairwise comparison
                   matrix w.r.t. Goal.
     alternatives = list of alternative names.
     alt_matrices  = list of alternatives pairwise comparison matrices
                     (upper triangular) w.r.t each main criterion.

   Attributes:
     Goal = Goal
     MC = list of main criteria
     MC_matrix = upper triangle of criteria pairwise comparison
                   matrix w.r.t. Goal.
     alternatives = list of alternative names
     alt_matrices = list of alternatives pairwise comparison matrix
                     (upper triangular) w.r.t each main criterion.
     n_MC  = number of main criteria
     n_alt = number of alternative

   Methods:
     model(): Get a summary of the AHP model
     solve(method='Algebra'): solve the model using method
     sensit(): perform sensitivity analysis and generate
               rainbow diagrams
```

[ ]:
```

```

9.2 Job Selection Problem with Sensitivity Analysis

Source: 9.4.3_Solve_Job_Selection_Problem_using_AHP3Lmodel_Class.ipynb

[1]:
```
""" Solve Job Selection Problem and perform Sensitivity Analysis
    using AHP3Lmodel Class """
from DecisionAnalysisPy import AHP3Lmodel
import numpy as np
```

[2]:
```
# Define your AHP model and data here
Goal = "Job Satisfaction"
main_criteria = ["Research",   "Growth",   "Benefits",
                 "Colleagues", "Location", "Reputation"]
# Upper triangle of criteria pairwise comparison matrix
main_criteria_matrix = np.array([1, 1, 4,  1,  1/2,
                                    2, 4,  1,  1/2,
                                       5,  3,  1/2,
                                          1/3, 1/3,
                                                1  ])

alternatives = ["Company A", "Company B", "Company C"]

# Upper triangles of alternatives pairwise comp matrix wrt each␣
↪criterion
alt_matrices  = [ np.array([1/4, 1/2,  3 ]),  # wrt Research
                  np.array([1/4, 1/5, 1/2]),  # wrt Growth
                  np.array([ 3,  1/3, 1/7]),  # wrt Benefits
                  np.array([1/3,  5,   7 ]),  # wrt Colleagues
                  np.array([ 1,   7,   7 ]),  # wrt Location
                  np.array([ 7,   9,   2 ]),  # wrt Reputation
                ]
# End of model definition and data
```

[3]:
```
# Create a 3-Level AHP model
JobSelect = AHP3Lmodel(Goal, main_criteria, main_criteria_matrix,
                       alternatives, alt_matrices)
```

[4]:
```
# Get a model structure and data
JobSelect.model()
```

```
Model Summary:
Goal: Job Satisfaction
Criteria:
  Number = 6
  ['Research', 'Growth', 'Benefits', 'Colleagues', 'Location',␣
↪'Reputation']

Pairwise comparison w.r.t. Goal Job Satisfaction:
[[ 1    1    1    4    1   1/2 ]
 [ 1    1    2    4    1   1/2 ]
 [ 1   1/2   1    5    3   1/2 ]
 [1/4  1/4  1/5   1   1/3  1/3 ]
 [ 1    1   1/3   3    1    1  ]
 [ 2    2    2    3    1    1  ]]

Alternatives:
  Number = 3
```

```
  ['Company A', 'Company B', 'Company C']

Pairwise comparison w.r.t criterion Research
[[ 1   1/4  1/2 ]
 [ 4    1    3  ]
 [ 2   1/3   1  ]]

Pairwise comparison w.r.t criterion Growth
[[ 1   1/4  1/5 ]
 [ 4    1   1/2 ]
 [ 5    2    1  ]]

Pairwise comparison w.r.t criterion Benefits
[[ 1    3   1/3 ]
 [1/3   1   1/7 ]
 [ 3    7    1  ]]

Pairwise comparison w.r.t criterion Colleagues
[[ 1   1/3   5  ]
 [ 3    1    7  ]
 [1/5  1/7   1  ]]

Pairwise comparison w.r.t criterion Location
[[ 1    1    7  ]
 [ 1    1    7  ]
 [1/7  1/7   1  ]]

Pairwise comparison w.r.t criterion Reputation
[[ 1    7    9  ]
 [1/7   1    2  ]
 [1/9  1/2   1  ]]
```

[5]:
```python
# Solve the model
results = JobSelect.solve(method='Algebra')
# "Power", "Algebra", "RGM", "ColsNorm", "GenEigen"
```

```
Model Summary:
  Goal: Job Satisfaction
  Criteria: ['Research', 'Growth', 'Benefits', 'Colleagues', 'Location',
'Reputation']
  Alternatives: ['Company A', 'Company B', 'Company C']

Criteria w.r.t. Goal Job Satisfaction:

[[ 1    1    1    4    1   1/2 ]
 [ 1    1    2    4    1   1/2 ]
 [ 1   1/2   1    5    3   1/2 ]
 [1/4  1/4  1/5   1   1/3  1/3 ]
```

```
 [ 1    1   1/3   3    1    1  ]
 [ 2    2    2    3    1    1  ]]
Lambda_max = 6.420344, CI= 0.084069, CR= 0.067797
Criteria Weights= [0.158408 0.189247 0.197997 0.04831  0.150245 0.255792]

Alternatives w.r.t. criterion Research
[[ 1   1/4  1/2 ]
 [ 4    1    3  ]
 [ 2   1/3   1  ]]
Lambda_max = 3.018295, CI= 0.009147, CR= 0.015771
Local Weights= [0.1365   0.625013 0.238487]

Alternatives w.r.t. criterion Growth
[[ 1   1/4  1/5 ]
 [ 4    1   1/2 ]
 [ 5    2    1  ]]
Lambda_max = 3.024595, CI= 0.012298, CR= 0.021203
Local Weights= [0.09739  0.333069 0.569541]

Alternatives w.r.t. criterion Benefits
[[ 1    3   1/3 ]
 [1/3   1   1/7 ]
 [ 3    7    1  ]]
Lambda_max = 3.007022, CI= 0.003511, CR= 0.006053
Local Weights= [0.242637 0.087946 0.669417]

Alternatives w.r.t. criterion Colleagues
[[ 1   1/3   5  ]
 [ 3    1    7  ]
 [1/5  1/7   1  ]]
Lambda_max = 3.064888, CI= 0.032444, CR= 0.055938
Local Weights= [0.278955 0.649118 0.071927]

Alternatives w.r.t. criterion Location
[[ 1    1    7  ]
 [ 1    1    7  ]
 [1/7  1/7   1  ]]
Lambda_max = 3.000000, CI= 0.000000, CR= 0.000000
Local Weights= [0.466667 0.466667 0.066667]

Alternatives w.r.t. criterion Reputation
[[ 1    7    9  ]
 [1/7   1    2  ]
 [1/9  1/2   1  ]]
Lambda_max = 3.021730, CI= 0.010865, CR= 0.018733
Local Weights= [0.792757 0.131221 0.076021]

Results:
```

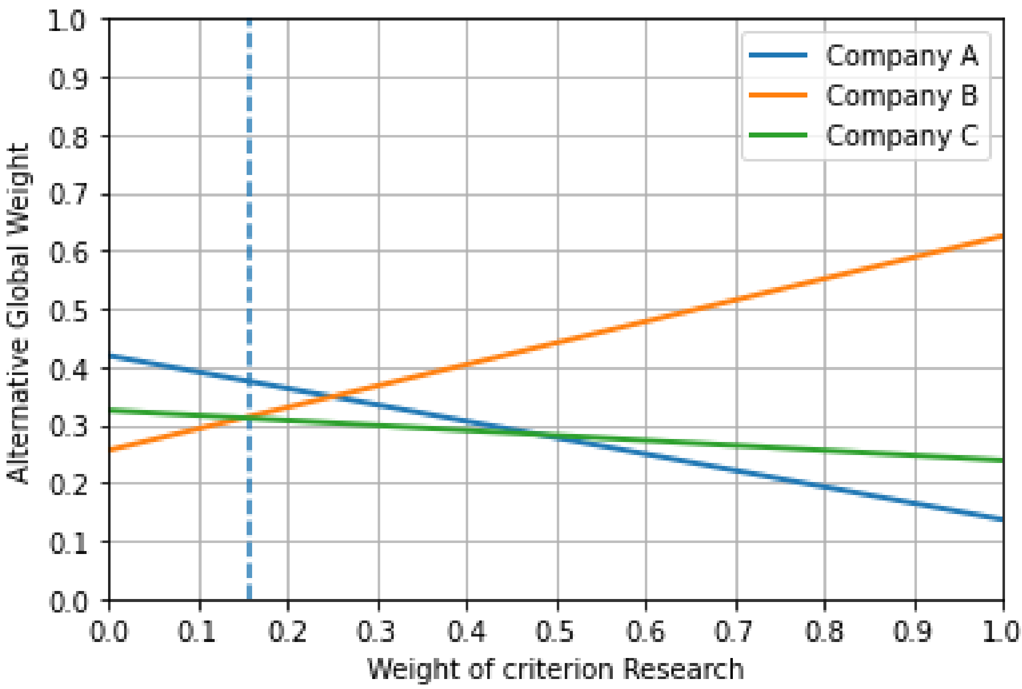
```
  Company A :0.374467
  Company B :0.314491
  Company C :0.311042

Sorted Results:
  Company A :0.374467
  Company B :0.314491
  Company C :0.311042
```

[6]:
```python
# Print just the alterative global weights
print(results)
```

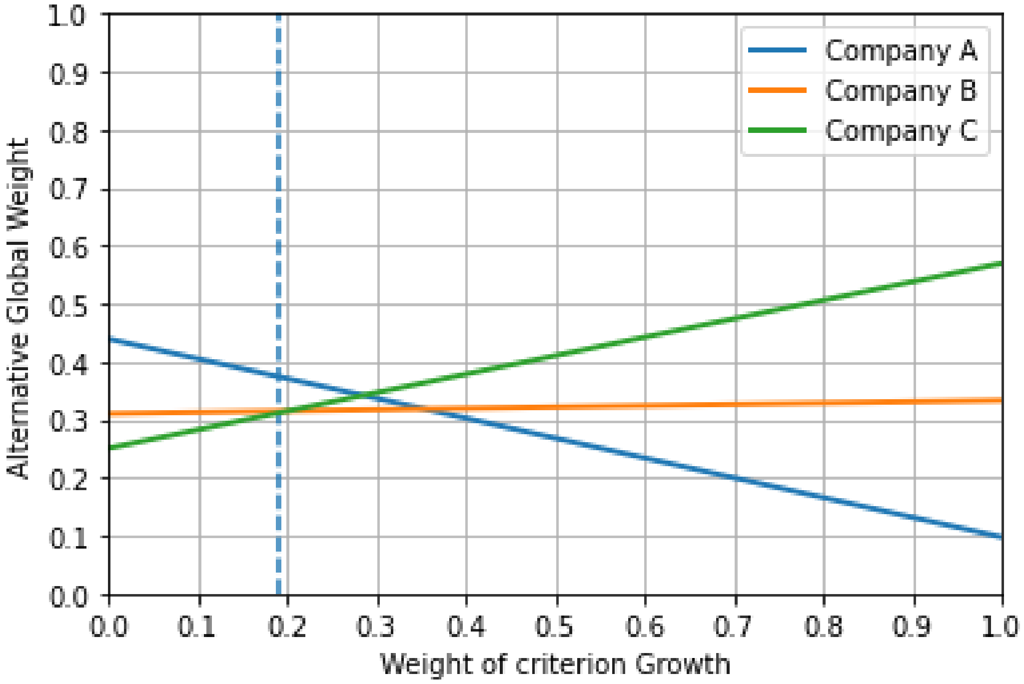
```
{'Company A': 0.3744666462485834, 'Company B': 0.3144914469034947,␣
 ↪'Company C':
0.31104190684792177}
```

[7]:
```python
# Perform Sensitivity Analysis
JobSelect.sensit()
```

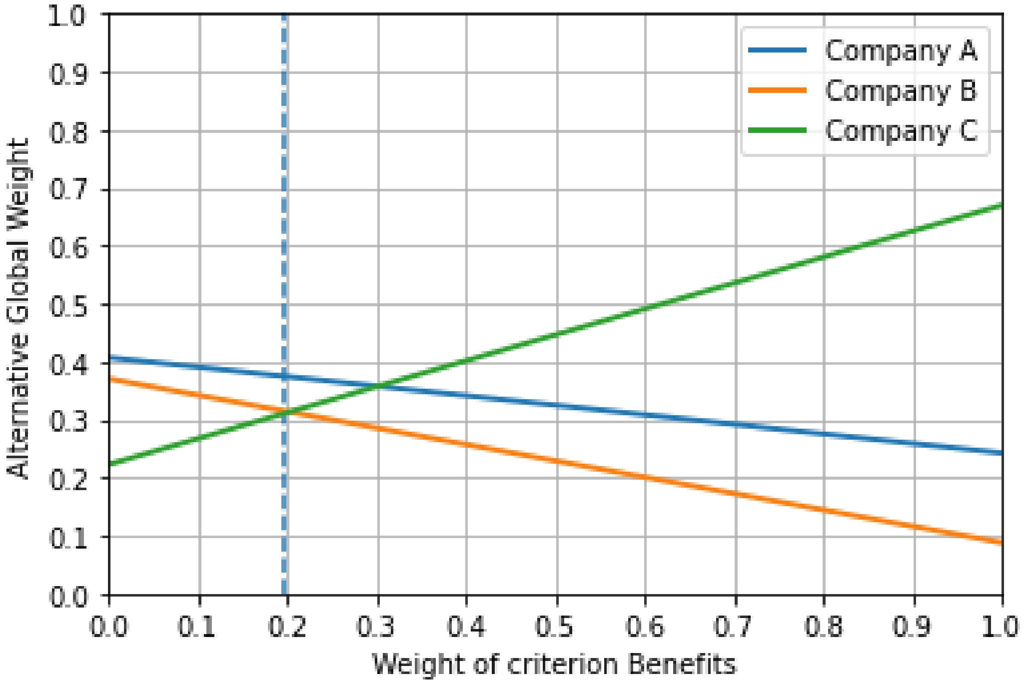
```
Sensivity Analysis:

Rainbow Diagram for changing weight of criterion Research
```



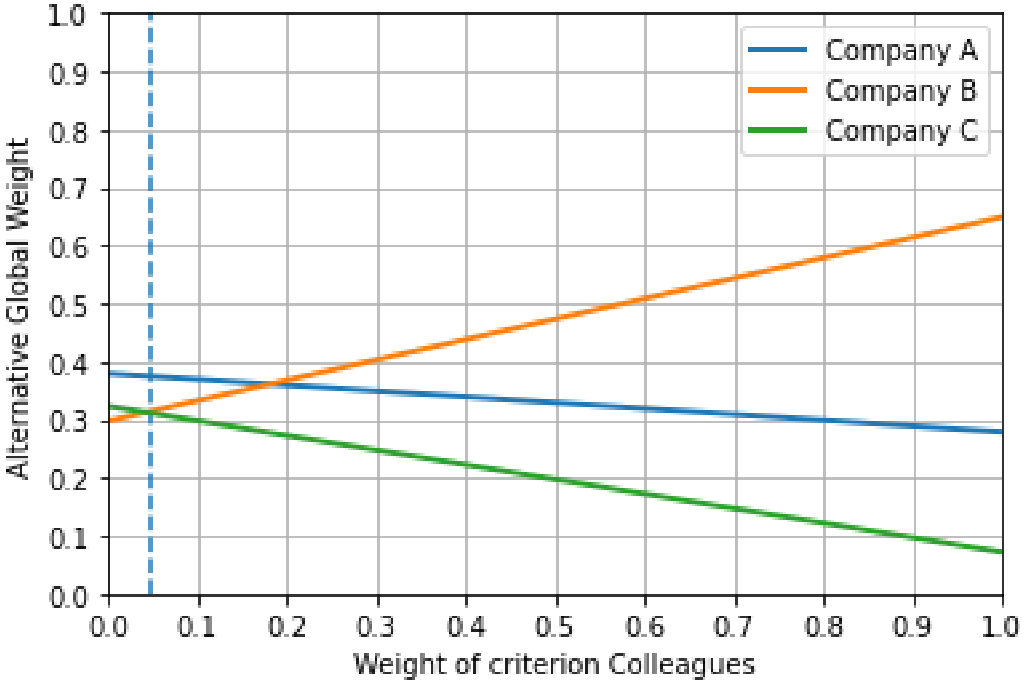

```
Rainbow Diagram for changing weight of criterion Growth
```



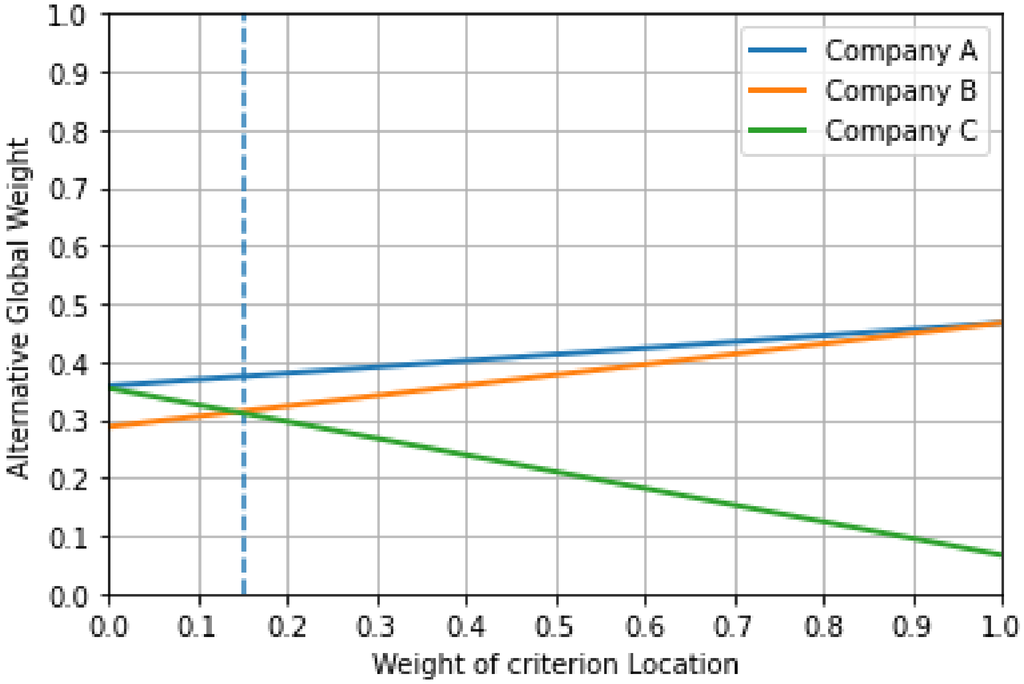

```
Rainbow Diagram for changing weight of criterion Benefits
```



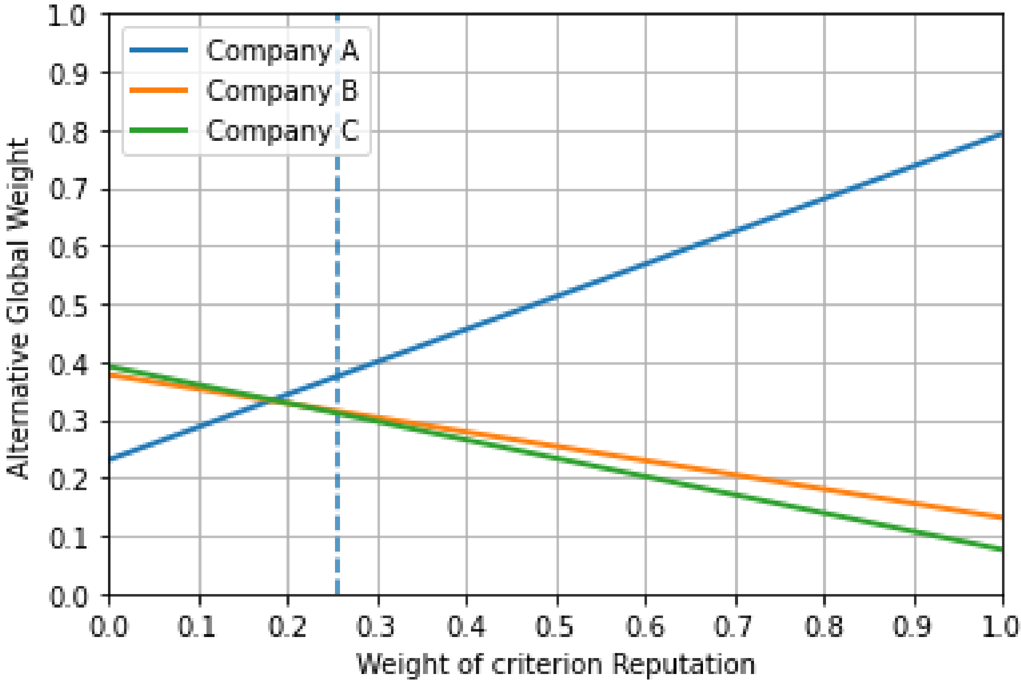

```
Rainbow Diagram for changing weight of criterion Colleagues
```



Rainbow Diagram for changing weight of criterion Location



Rainbow Diagram for changing weight of criterion Reputation



[ ]:

```

```

9.3 Problem 9.1: Computer Selection Problem

Source: 9_Problem_9.1_Computer_Selection_Problem.ipynb

[1]:
```
""" Problem 9.1: Computer Selection Problem"""
from DecisionAnalysisPy import AHP3Lmodel
import numpy as np
```

[2]:
```
Goal = "Best Computer"
main_criteria = ["Cost", "User-friendiness", "Software availability"]

# Upper triangle of criteria pairwise comparison matrix
main_criteria_matrix = np.array([1/4, 1/5, 1/2 ])

alternatives = ["Computer 1", "Computer 2", "Computer 3"]
# Upper triangles of alternatives pairwise comp matrix wrt each␣
↪criterion
alt_matrices = [ np.array([3,    5,    2 ]),  # wrt Cost
                 np.array([1/3, 1/2,   5 ]),  # wrt User-friendiness
                 np.array([1/3, 1/7, 1/5]) ] # wrt Software␣
↪availability

# End of model definition and data
```

[3]:
```
# Create a 3-Level AHHP model
CompSelect = AHP3Lmodel(Goal, main_criteria, main_criteria_matrix,
                        alternatives, alt_matrices)
```

[4]:
```
# Get model structure and data
CompSelect.model()
```

```
Model Summary:
Goal: Best Computer
Criteria:
  Number = 3
  ['Cost', 'User-friendiness', 'Software availability']

Pairwise comparison w.r.t. Goal Best Computer:
[[ 1   1/4  1/5 ]
 [ 4    1   1/2 ]
 [ 5    2    1  ]]

Alternatives:
  Number = 3
  ['Computer 1', 'Computer 2', 'Computer 3']

Pairwise comparison w.r.t criterion Cost
[[ 1    3    5  ]
 [1/3   1    2  ]
 [1/5  1/2   1  ]]

Pairwise comparison w.r.t criterion User-friendiness
[[ 1   1/3  1/2 ]
 [ 3    1    5  ]
 [ 2   1/5   1  ]]

Pairwise comparison w.r.t criterion Software availability
[[ 1   1/3  1/7 ]
 [ 3    1   1/5 ]
 [ 7    5    1  ]]
```

[5]:
```
# Solve the model
result = CompSelect.solve(method='Power')
# "Power", "Algebra", "RGM", "ColsNorm", "GenEigen"
```

```
Model Summary:
  Goal: Best Computer
  Criteria: ['Cost', 'User-friendiness', 'Software availability']
  Alternatives: ['Computer 1', 'Computer 2', 'Computer 3']

Criteria w.r.t. Goal Best Computer:
```

```
[[ 1   1/4  1/5 ]
 [ 4    1   1/2 ]
 [ 5    2    1  ]]
Lambda_max = 3.024595, CI= 0.012298, CR= 0.021203
Criteria Weights= [0.09739  0.333069 0.569541]

Alternatives w.r.t. criterion Cost
[[ 1    3    5  ]
 [1/3   1    2  ]
 [1/5  1/2   1  ]]
Lambda_max = 3.003695, CI= 0.001847, CR= 0.003185
Local Weights= [0.648329 0.229651 0.12202 ]
Warning: CR > 0.1

Alternatives w.r.t. criterion User-friendiness
[[ 1   1/3  1/2 ]
 [ 3    1    5  ]
 [ 2   1/5   1  ]]
Lambda_max = 3.163235, CI= 0.081617, CR= 0.140719
Local Weights= [0.146622 0.657071 0.196307]

Alternatives w.r.t. criterion Software availability
[[ 1   1/3  1/7 ]
 [ 3    1   1/5 ]
 [ 7    5    1  ]]
Lambda_max = 3.064888, CI= 0.032444, CR= 0.055938
Local Weights= [0.080961 0.188394 0.730645]

Results:
  Computer 1 :0.158087
  Computer 2 :0.348514
  Computer 3 :0.493399

Sorted Results:
  Computer 3 :0.493399
  Computer 2 :0.348514
  Computer 1 :0.158087
```

[6]:

```
# print the alternative global weights
print(result)
```

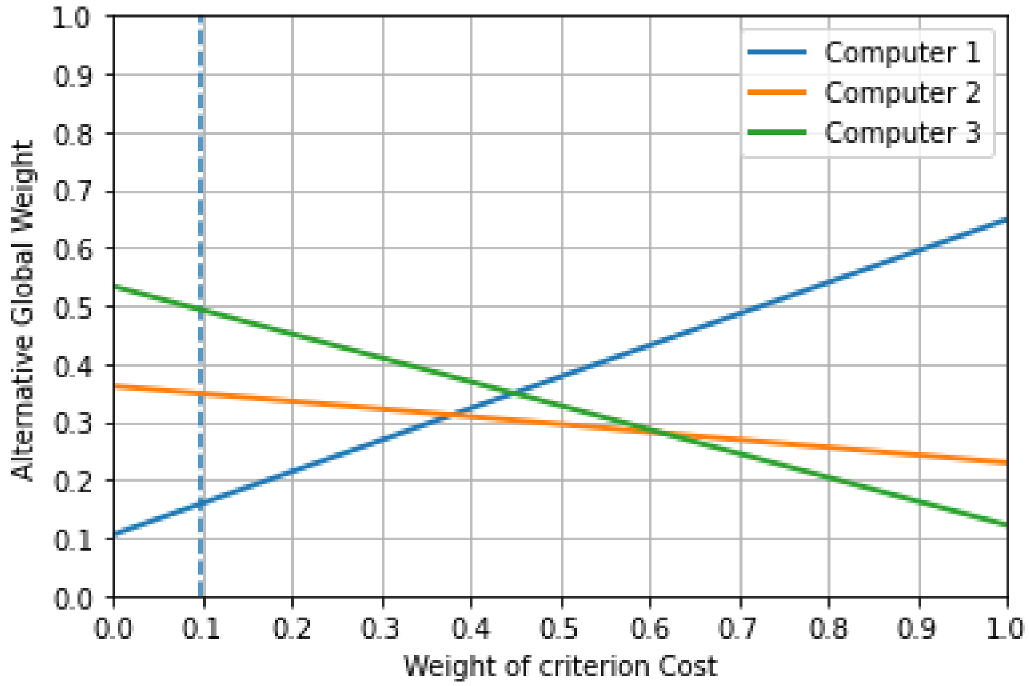
```
{'Computer 1': 0.1580867247845747, 'Computer 2': 0.3485141329551402,␣
 ↪'Computer
3': 0.4933991422602852}
```

[7]:

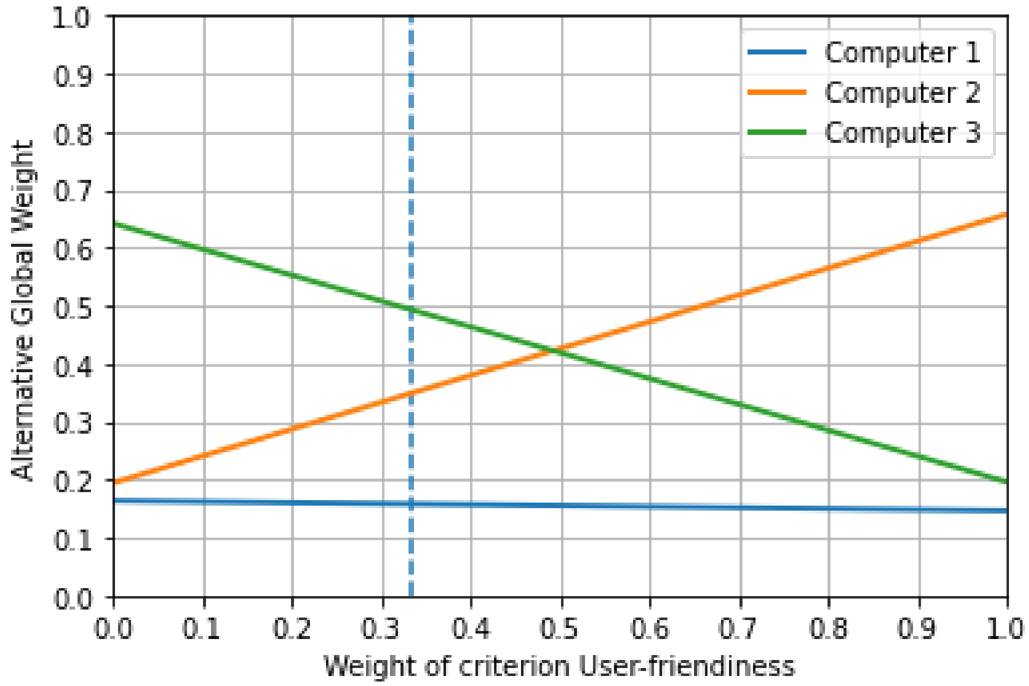
```
# Perform sensitivity analysis
CompSelect.sensit()
```

Sensivity Analysis:

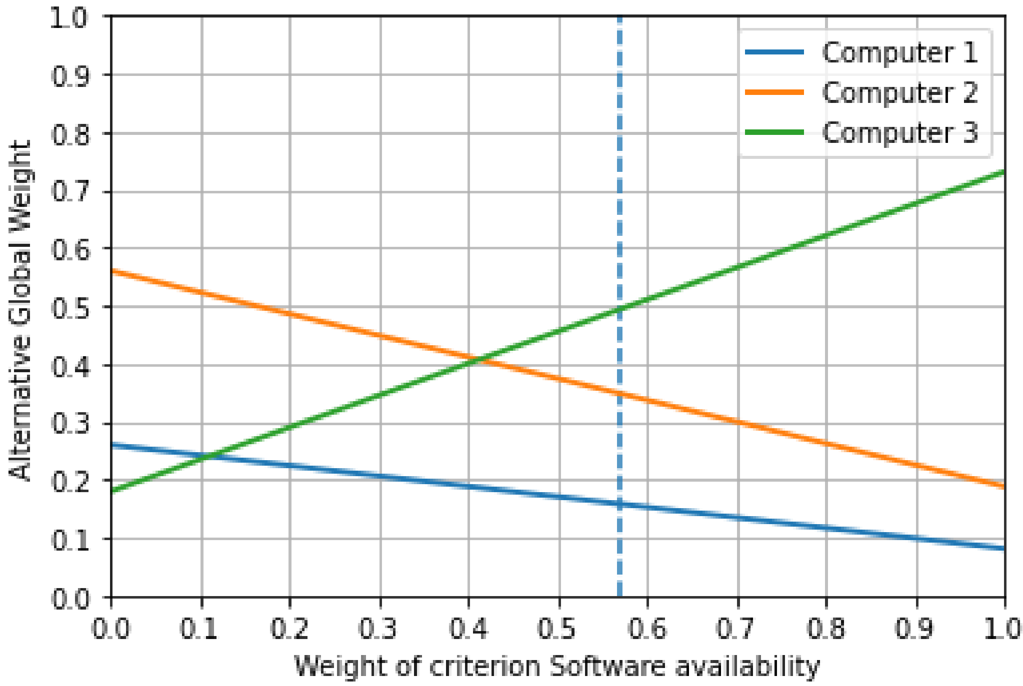
Rainbow Diagram for changing weight of criterion Cost



Rainbow Diagram for changing weight of criterion User-friendiness



```
Rainbow Diagram for changing weight of criterion Software availability
```




```
[ ]:
```

9.4 Problem 9.3: Rank Reversal Problem

Source 9_Problem_9.3_Rank_Reversal_Problem.ipynb

```
[1]: """ Problem 9.3: Rank Reversal Problem with 2 and 3 Alternatives"""
from DecisionAnalysisPy import AHP3Lmodel
import numpy as np
```

```
[2]: # Set up the common Goal and Criteria
Goal = "Best Investment"
main_criteria = ["Expected Return", "Degree of Risk" ]

# Upper triangle of criteria pairwise comparison matrix
main_criteria_matrix = np.array([ 1 ])
```

```
[3]: """ When there are 2 alternatives """
alternatives_2A = ["Investment 1", "Investment 2" ]
# Upper triangles of alternatives pairwise comp matrix wrt each␣
↪criterion
alt_matrices_2A  = [ np.array([1/2]),    # wrt Expected Return
```

```
                        np.array([ 3 ]) ]  # wrt Degree of Risk
# Create a 3-Level AHHP model with 2 alternatives
AHP_2A = AHP3Lmodel(Goal, main_criteria, main_criteria_matrix,
                    alternatives_2A, alt_matrices_2A)
# AHP_2A.model()
# Solve it
result_2A = AHP_2A.solve(method='Algebra')
# AHP_2A.sensit()
```

```
Model Summary:
  Goal: Best Investment
  Criteria: ['Expected Return', 'Degree of Risk']
  Alternatives: ['Investment 1', 'Investment 2']

Criteria w.r.t. Goal Best Investment:

[[ 1    1  ]
 [ 1    1  ]]
Lambda_max = 2.000000, CI= 0.000000, CR= 0.000000
Criteria Weights= [0.5 0.5]

Alternatives w.r.t. criterion Expected Return
[[ 1   1/2 ]
 [ 2    1  ]]
Lambda_max = 2.000000, CI= 0.000000, CR= 0.000000
Local Weights= [0.333333 0.666667]

Alternatives w.r.t. criterion Degree of Risk
[[ 1    3  ]
 [1/3   1  ]]
Lambda_max = 2.000000, CI= 0.000000, CR= 0.000000
Local Weights= [0.75 0.25]

Results:
  Investment 1 :0.541667
  Investment 2 :0.458333

Sorted Results:
  Investment 1 :0.541667
  Investment 2 :0.458333
```

[4]:
```
""" When there are 3 alternatives """
alternatives_3A = ["Investment 1", "Investment 2", "Investment 3" ]
# Upper triangles of alternatives pairwise comp matrix wrt each␣
 ↪criterion
alt_matrices_3A  = [ np.array([1/2,  4,   8]),     # wrt Expected␣
 ↪Return
```

```
                          np.array([ 3,  1/2, 1/6 ]) ]  # wrt Degree of Risk
# Create a 3-Level AHHP model with 3 alternatives
AHP_3A = AHP3Lmodel(Goal, main_criteria, main_criteria_matrix,
                         alternatives_3A, alt_matrices_3A)
# AHP_3A.model()
# Solve it
result_3A = AHP_3A.solve(method='Algebra')
```

```
Model Summary:
  Goal: Best Investment
  Criteria: ['Expected Return', 'Degree of Risk']
  Alternatives: ['Investment 1', 'Investment 2', 'Investment 3']

Criteria w.r.t. Goal Best Investment:

[[ 1    1  ]
 [ 1    1  ]]
Lambda_max = 2.000000, CI= 0.000000, CR= 0.000000
Criteria Weights= [0.5 0.5]

Alternatives w.r.t. criterion Expected Return
[[ 1   1/2   4  ]
 [ 2    1    8  ]
 [1/4  1/8   1  ]]
Lambda_max = 3.000000, CI= 0.000000, CR= 0.000000
Local Weights= [0.307692 0.615385 0.076923]

Alternatives w.r.t. criterion Degree of Risk
[[ 1    3   1/2 ]
 [1/3   1   1/6 ]
 [ 2    6    1  ]]
Lambda_max = 3.000000, CI= 0.000000, CR= 0.000000
Local Weights= [0.3 0.1 0.6]

Results:
  Investment 1 :0.303846
  Investment 2 :0.357692
  Investment 3 :0.338462

Sorted Results:
  Investment 2 :0.357692
  Investment 3 :0.338462
  Investment 1 :0.303846
```

[5]:
```
""" Compare the two results """
print("\nCompare the results:")
print("\n  When are are 2 Alternatives:")
```

```
for k, v in sorted(result_2A.items(), key=lambda x: x[1], reverse=True):
    print(f"    {k} : {v:.6f}")

print("\n  When are are 3 Alternatives:")
for k, v in sorted(result_3A.items(), key=lambda x: x[1], reverse=True):
    print(f"    {k} : {v:.6f}")
```

```
Compare the results:

  When are are 2 Alternatives:
    Investment 1 : 0.541667
    Investment 2 : 0.458333

  When are are 3 Alternatives:
    Investment 2 : 0.357692
    Investment 3 : 0.338462
    Investment 1 : 0.303846
```

[ ]:

```

```

9.5 Problem 10.4: System Selection Problem AHP Model

Source: 10_Problem_10.4_System_Selection_Problem.ipynb

[1]:

```
""" Problem 10.4: System Selection Problem """
from DecisionAnalysisPy import AHP3Lmodel
import numpy as np
```

[2]:

```
Goal = "Best System"
main_criteria =["Human Productivity", "Economics", "Design",␣
 ↪"Operations"]

# Upper triangle of criteria pairwise comparison matrix
main_criteria_matrix = np.array([ 3, 3, 7,
                                     2, 5,
                                        7 ])

alternatives = ["System A", "System B", "System C" ]
# Upper triangles of alternatives pairwise comp matrix wrt each␣
 ↪criterion
alt_matrices  = [ np.array([ 3,    5,    2 ]),   # wrt Human Productivity
                  np.array([1/3, 1/2,   3 ]),   # wrt Economics
                  np.array([1/2, 1/7, 1/5]),   # wrt Design
                  np.array([ 3,   1/5, 1/9]),   # wrt Operations
                ]
```

[3]:
```python
# Create a 3-Level AHP model
SysSelect = AHP3Lmodel(Goal, main_criteria, main_criteria_matrix,
                       alternatives, alt_matrices)
```

[4]:
```python
# Get model structure and data
SysSelect.model()
```

```
Model Summary:
Goal: Best System
Criteria:
  Number = 4
  ['Human Productivity', 'Economics', 'Design', 'Operations']

Pairwise comparison w.r.t. Goal Best System:
[[ 1    3    3    7  ]
 [1/3   1    2    5  ]
 [1/3  1/2   1    7  ]
 [1/7  1/5  1/7   1  ]]

Alternatives:
  Number = 3
  ['System A', 'System B', 'System C']

Pairwise comparison w.r.t criterion Human Productivity
[[ 1    3    5  ]
 [1/3   1    2  ]
 [1/5  1/2   1  ]]

Pairwise comparison w.r.t criterion Economics
[[ 1   1/3  1/2 ]
 [ 3    1    3  ]
 [ 2   1/3   1  ]]

Pairwise comparison w.r.t criterion Design
[[ 1   1/2  1/7 ]
 [ 2    1   1/5 ]
 [ 7    5    1  ]]

Pairwise comparison w.r.t criterion Operations
[[ 1    3   1/5 ]
 [1/3   1   1/9 ]
 [ 5    9    1  ]]
```

[5]:
```python
# Solve the model
results = SysSelect.solve(method='Algebra')
```

```
Model Summary:
  Goal: Best System
```

```
  Criteria: ['Human Productivity', 'Economics', 'Design', 'Operations']
  Alternatives: ['System A', 'System B', 'System C']

Criteria w.r.t. Goal Best System:

[[ 1    3    3    7  ]
 [1/3   1    2    5  ]
 [1/3  1/2   1    7  ]
 [1/7  1/5  1/7   1  ]]
Lambda_max = 4.212088, CI= 0.070696, CR= 0.078551
Criteria Weights= [0.513052 0.246592 0.193575 0.046781]

Alternatives w.r.t. criterion Human Productivity
[[ 1    3    5  ]
 [1/3   1    2  ]
 [1/5  1/2   1  ]]
Lambda_max = 3.003695, CI= 0.001847, CR= 0.003185
Local Weights= [0.648329 0.229651 0.12202 ]

Alternatives w.r.t. criterion Economics
[[ 1   1/3  1/2 ]
 [ 3    1    3  ]
 [ 2   1/3   1  ]]
Lambda_max = 3.053622, CI= 0.026811, CR= 0.046225
Local Weights= [0.157056 0.593634 0.249311]

Alternatives w.r.t. criterion Design
[[ 1   1/2  1/7 ]
 [ 2    1   1/5 ]
 [ 7    5    1  ]]
Lambda_max = 3.014152, CI= 0.007076, CR= 0.012200
Local Weights= [0.093813 0.166593 0.739594]

Alternatives w.r.t. criterion Operations
[[ 1    3   1/5 ]
 [1/3   1   1/9 ]
 [ 5    9    1  ]]
Lambda_max = 3.029064, CI= 0.014532, CR= 0.025055
Local Weights= [0.178178 0.070418 0.751405]

Results:
  System A :0.397850
  System B :0.299750
  System C :0.302399

Sorted Results:
  System A :0.397850
  System C :0.302399
```

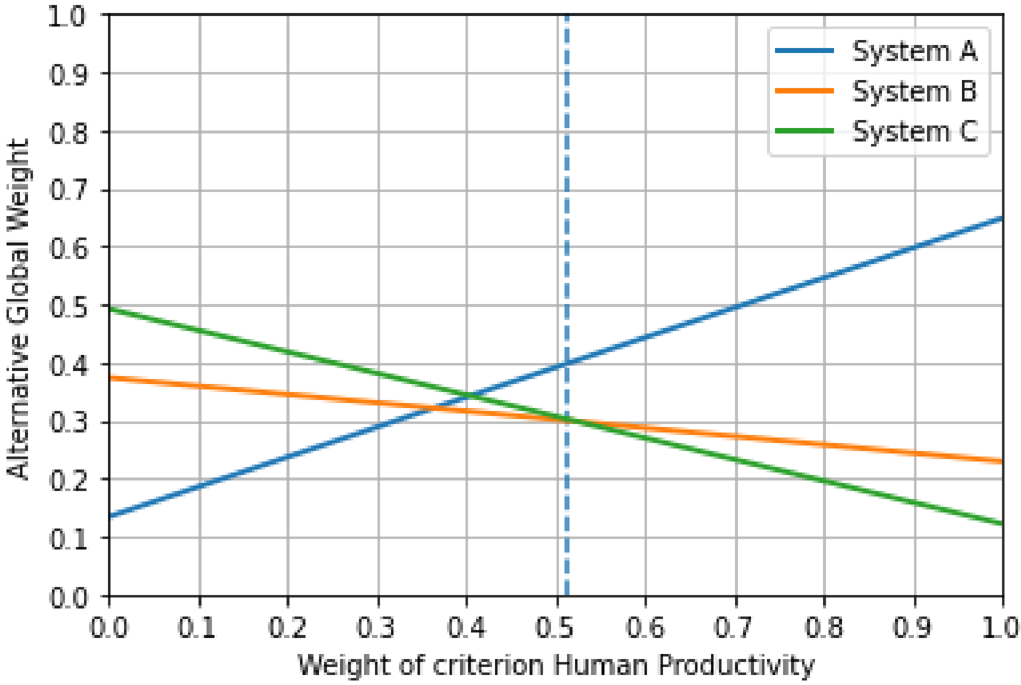
```
System B :0.299750
```

[6]:
```
# Print the alternative global weights
print(results)
```

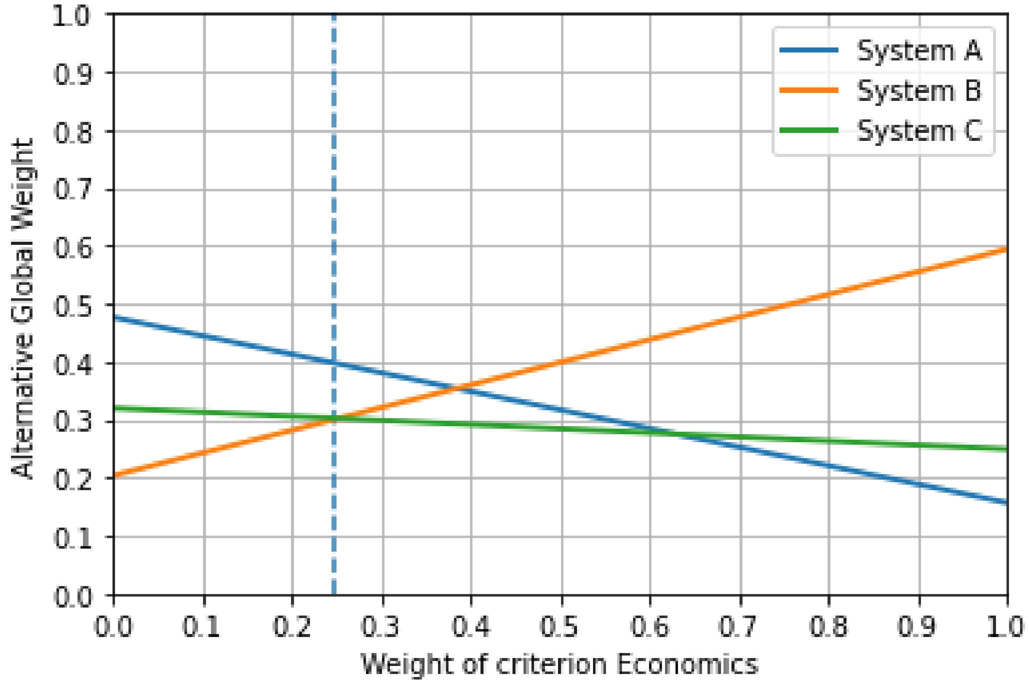
```
{'System A': 0.3978503249070714, 'System B': 0.29975039656886754,
 'System C':
0.30239927852406107}
```

[7]:
```
# Performance sensitivity analysis
SysSelect.sensit()
```

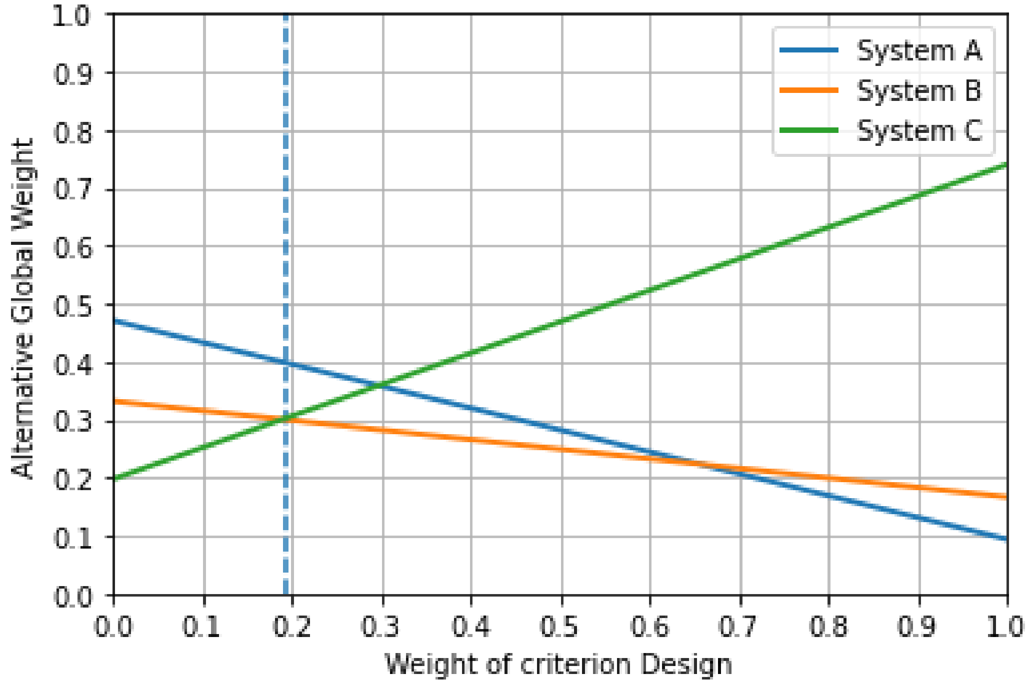
```
Sensivity Analysis:

Rainbow Diagram for changing weight of criterion Human Productivity
```



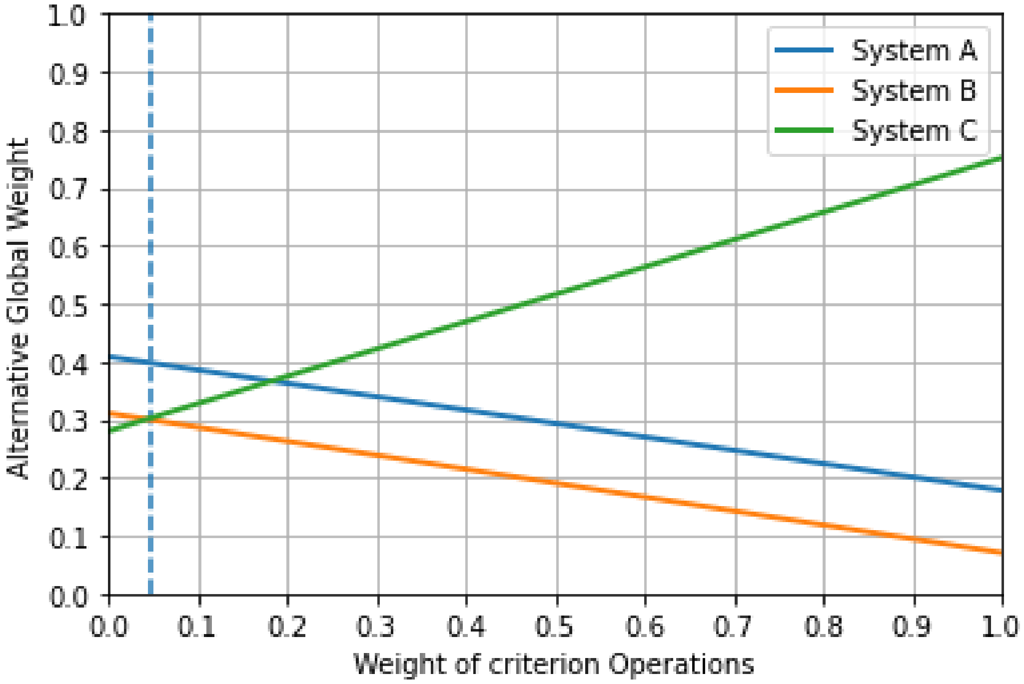

```
Rainbow Diagram for changing weight of criterion Economics
```



Rainbow Diagram for changing weight of criterion Design



Rainbow Diagram for changing weight of criterion Operations


System A
System B
System C
Alternative Global Weight
Weight of criterion Operations
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

```
[ ]:
```

10 Class AHP4Lmodel

10.1 Documentation

[1]:
```python
from DecisionAnalysisPy import AHP4Lmodel
```

[2]:
```python
print(AHP4Lmodel.__doc__)
```

```
Class for 4-Level AHP Models
   AHP4Lmodel(Goal, MC, MC_matrix, SC, SC_matrices,
                    alternatives, alt_matrices):
   Parameters:
     Goal = Goal
     MC = list of main criteria
     MC_matrix = pairwise comparison matrix of main criteria
                 (upper triangle) w.r.t. Goal.
     SC = list of lists of sub-criteria w.r.t. each main criterion
     SC_matrices = list of lists of pairwise comparison matrices
                   (upper triangle) w.r.t each main criterion
     alternatives = list of alternative names
     alt_matrices = list of lists of alternative pairwise comparison
                matrices (upper triangle) w.r.t. each sub-criterion

   Attributes:
     goal = Goal
     MC = list of main criteria
     n_MC  = number of main criteria
     MC_matrix = AHPmatrix for main criteria
     SC = list of lists of sub-criteria w.r.t. each main criterion
     SC_matrices = list of lists of AHPmatrix w.r.t each main criterion
     alternatives = list of alternative names
     alt_matrices = list of lists of alternative AHPmatrix w.r.t. each
                   sub-criterion
     n_alt = number of alternatives

     MC_w = array of main criteria weights
     SC_weights = list of arrays of sub-criteria weights
     alt_weights = list of arrays of alternative weights

   Methods:
     model() : show model summary
     solve(method='Algebra'): solve the AHP model using method
     sensit(ymax=None): perform sensitivity analysis.
                        ymax = upper limit of rainbow diagrams to plot
```

[ ]:

10.2 Job Selection Problem with SubCriteria and Sensitivity Analysis

Source: 9.5_Solve_Job_Selection_Problem_with_SubCriteria_using_AHP4Lmodel_Class.ipynb

[1]:
```
""" Solve Job Selection Problem with Sub-criteria using AHP4Lmodel␣
↪Class
    This a asymetric 4-level hierarchy problem """
from DecisionAnalysisPy import AHP4Lmodel
import numpy as np
```

[2]:
```
Goal = "Job Satisfaction"
alternatives = ["Company A", "Company B", "Company C"]

main_criteria = ["Research",   "Growth",    "Benefits",
                 "Colleagues", "Location", "Reputation" ]

main_criteria_matrix = np.array([1, 1, 4,  1,  1/2,
                                    2, 4,  1,  1/2,
                                       5,  3,  1/2,
                                          1/3, 1/3,
                                                1  ] )
```

[3]:
```
# Containers for model data
sub_criteria = []
sub_criteria_matrices = []
alt_matrices = []
```

[4]:
```
# Main Criterion 1: "Research"
# List of subcriteria for Criterion 1
sub_criteria.append(None)
# Pairwise comparison of subcriteria for Criterion 1
sub_criteria_matrices.append(None)
# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_matrices.append([np.array([1/4, 1/2, 3])])
```

[5]:
```
# Main Criterion 2: "Growth"
# List of subcriteria for Criterion 2
sub_criteria.append(["Short-term growth", "Long-term growth" ])
# Pairwise comparison of subcriteria for Criterion 2
sub_criteria_matrices.append(np.array([ 1/3 ]))
# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_matrices.append([np.array([1/3, 1/7, 1/3 ]),
                     np.array([ 3,   5,   2  ])])
```

[6]:
```
# Main Criterion 3: "Benefits"
# List of subcriteria for Criterion 3
sub_criteria.append(None)
```

```
# Pairwise comparison of subcriteria for Criterion 3
sub_criteria_matrices.append(None)
# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_matrices.append([np.array([3, 1/3, 1/7 ])])
```

[7]:
```
# Main Criterion 4: "Colleagues"
# List of subcriteria for Criterion 4
sub_criteria.append(None)
# Pairwise comparison of subcriteria for Criterion 4
sub_criteria_matrices.append(None)
# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_matrices.append([np.array([1/3, 5, 7 ])])
```

[8]:
```
# Main Criterion 5: "Location"
# List of subcriteria for Criterion 5
sub_criteria.append(None)
# Pairwise comparison of subcriteria for Criterion 5
sub_criteria_matrices.append(None)
# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_matrices.append([np.array([1, 7, 7 ])])
```

[9]:
```
# Main Criterion 6: "Reputation"
# List of subcriteria for Criterion 6
sub_criteria.append(None)
# Pairwise comparison of subcriteria for Criterion 5
sub_criteria_matrices.append(None)
# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_matrices.append([np.array([7, 9, 2 ])])

# End of model definition and data
```

[10]:
```
# Create a 4-Level AHP model
JobSelect_SC = AHP4Lmodel(Goal, main_criteria, main_criteria_matrix,
                        sub_criteria, sub_criteria_matrices,
                        alternatives, alt_matrices)
```

[11]:
```
# Get model structure and data
JobSelect_SC.model()
```

```
Goal: Job Satisfaction

Number of main criteria = 6
Main Criteria: ['Research', 'Growth', 'Benefits', 'Colleagues',␣
 ↪'Location',
'Reputation']

Pairwise comparison w.r.t. Goal Job Satisfaction:
```

```
[[ 1    1    1    4    1   1/2 ]
 [ 1    1    2    4    1   1/2 ]
 [ 1   1/2   1    5    3   1/2 ]
 [1/4  1/4  1/5   1   1/3  1/3 ]
 [ 1    1   1/3   3    1    1  ]
 [ 2    2    2    3    1    1  ]]

Main Criteron 1: Research
Criterion Research has no sub-criterion

Number of alternatives = 3
['Company A', 'Company B', 'Company C']
Pairwise comparison w.r.t. Research
[[ 1   1/4  1/2 ]
 [ 4    1    3  ]
 [ 2   1/3   1  ]]

Main Criteron 2: Growth
  Number of sub-criteria = 2
  ['Short-term growth', 'Long-term growth']
Pairwise comparison w.r.t {self.MC[cr]}:
[[ 1   1/3 ]
 [ 3    1  ]]

Number of alternatives = 3
['Company A', 'Company B', 'Company C']

Pairwise comparison w.r.t.  Short-term growth
[[ 1   1/3  1/7 ]
 [ 3    1   1/3 ]
 [ 7    3    1  ]]

Number of alternatives = 3
['Company A', 'Company B', 'Company C']

Pairwise comparison w.r.t.  Long-term growth
[[ 1    3    5  ]
 [1/3   1    2  ]
 [1/5  1/2   1  ]]

Main Criteron 3: Benefits
Criterion Benefits has no sub-criterion

Number of alternatives = 3
['Company A', 'Company B', 'Company C']
Pairwise comparison w.r.t. Benefits
[[ 1    3   1/3 ]
 [1/3   1   1/7 ]
```

```
 [ 3    7    1  ]]

Main Criteron 4: Colleagues
Criterion Colleagues has no sub-criterion

Number of alternatives = 3
['Company A', 'Company B', 'Company C']
Pairwise comparison w.r.t. Colleagues
[[ 1   1/3   5  ]
 [ 3    1    7  ]
 [1/5  1/7   1  ]]

Main Criteron 5: Location
Criterion Location has no sub-criterion

Number of alternatives = 3
['Company A', 'Company B', 'Company C']
Pairwise comparison w.r.t. Location
[[ 1    1    7  ]
 [ 1    1    7  ]
 [1/7  1/7   1  ]]

Main Criteron 6: Reputation
Criterion Reputation has no sub-criterion

Number of alternatives = 3
['Company A', 'Company B', 'Company C']
Pairwise comparison w.r.t. Reputation
[[ 1    7    9  ]
 [1/7   1    2  ]
 [1/9  1/2   1  ]]
```

[12]:
```
# Solve the model
result = JobSelect_SC.solve(method="Algebra")
```

```
Goal: Job Satisfaction
Alternatives: ['Company A', 'Company B', 'Company C']
Main Criteria: ['Research', 'Growth', 'Benefits', 'Colleagues',
 ↪'Location',
'Reputation']

Pairwise comparison of main criteria w.r.t. Goal Job Satisfaction:
[[ 1    1    1    4    1   1/2 ]
 [ 1    1    2    4    1   1/2 ]
 [ 1   1/2   1    5    3   1/2 ]
 [1/4  1/4  1/5   1   1/3  1/3 ]
 [ 1    1   1/3   3    1    1  ]
```

```
 [ 2    2    2    3    1    1  ]]
Lambda = 6.420344, CI= 0.084069, CR= 0.067797
Main criteria weights= [0.158408 0.189247 0.197997 0.04831  0.150245 0.
 ↪255792]
Inside None
[[ 1   1/4  1/2 ]
 [ 4    1    3  ]
 [ 2   1/3   1  ]]
Lambda = 3.018295, CI= 0.009147, CR= 0.015771
Alternative weights= [0.1365   0.625013 0.238487]
Pairwise comparison of sub-criteria for Growth:
[[ 1   1/3 ]
 [ 3    1  ]]
Lambda = 2.000000, CI= 0.000000, CR= 0.000000
Sub-citeria weights= [0.25 0.75]
[[ 1   1/3  1/7 ]
 [ 3    1   1/3 ]
 [ 7    3    1  ]]
Lambda = 3.007022, CI= 0.003511, CR= 0.006053
Alternative weights= [0.087946 0.242637 0.669417]
[[ 1    3    5  ]
 [1/3   1    2  ]
 [1/5  1/2   1  ]]
Lambda = 3.003695, CI= 0.001847, CR= 0.003185
Alternative weights= [0.648329 0.229651 0.12202 ]
Inside None
[[ 1    3   1/3 ]
 [1/3   1   1/7 ]
 [ 3    7    1  ]]
Lambda = 3.007022, CI= 0.003511, CR= 0.006053
Alternative weights= [0.242637 0.087946 0.669417]
Inside None
[[ 1   1/3   5  ]
 [ 3    1    7  ]
 [1/5  1/7   1  ]]
Lambda = 3.064888, CI= 0.032444, CR= 0.055938
Alternative weights= [0.278955 0.649118 0.071927]
Inside None
[[ 1    1    7  ]
 [ 1    1    7  ]
 [1/7  1/7   1  ]]
Lambda = 3.000000, CI= 0.000000, CR= 0.000000
Alternative weights= [0.466667 0.466667 0.066667]
Inside None
[[ 1    7    9  ]
 [1/7   1    2  ]
 [1/9  1/2   1  ]]
Lambda = 3.021730, CI= 0.010865, CR= 0.018733
```

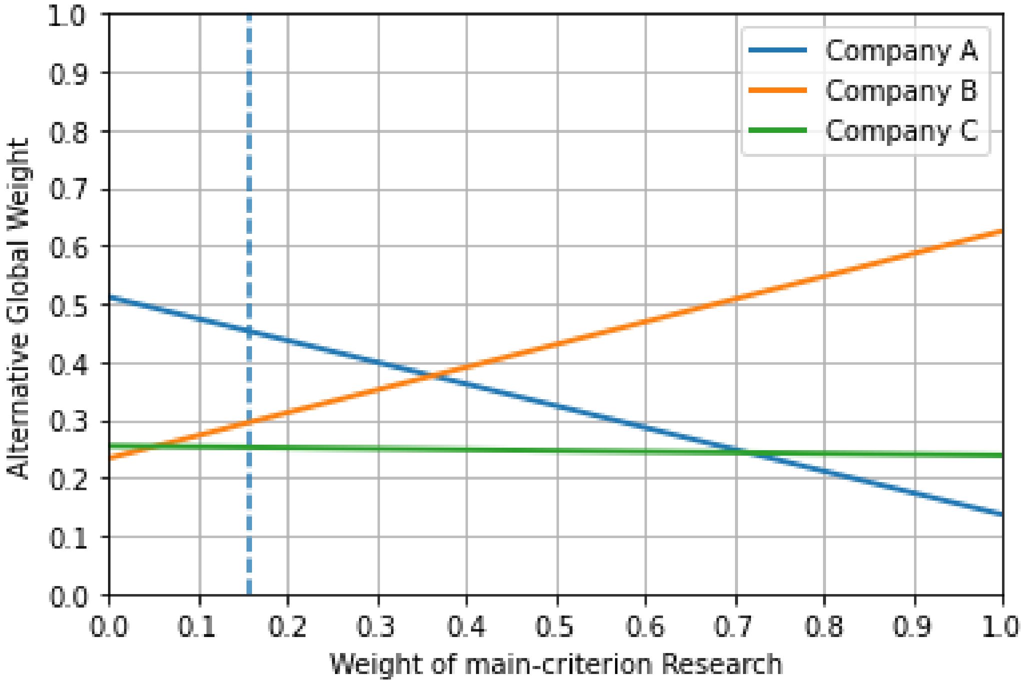
```
Alternative weights= [0.792757 0.131221 0.076021]

Results:
  Company A : 0.452218
  Company B : 0.295534
  Company C : 0.252248

Sorted Results:
  Company A : 0.452218
  Company B : 0.295534
  Company C : 0.252248
```

[13]:
```python
# Print the alternative global weights
print(result)
```

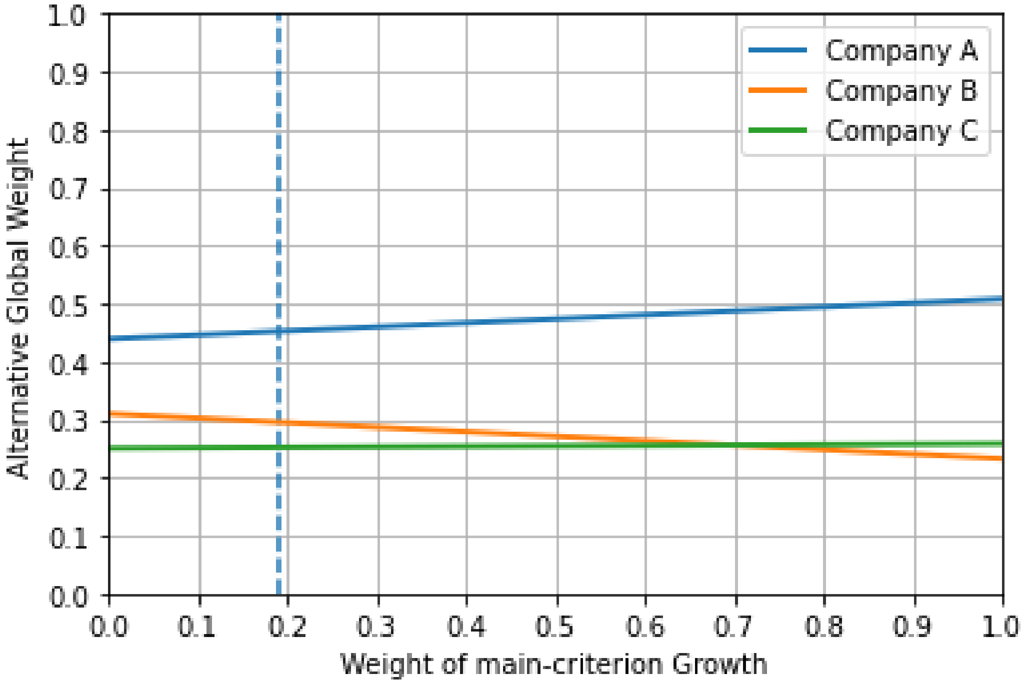
```
{'Company A': 0.45221759857347116, 'Company B': 0.2955341669753073,␣
↪'Company C':
0.25224823445122135}
```

[14]:
```python
# Perform sensitivity analsyis
JobSelect_SC.sensit(ymax=1, ystep=0.1)
```

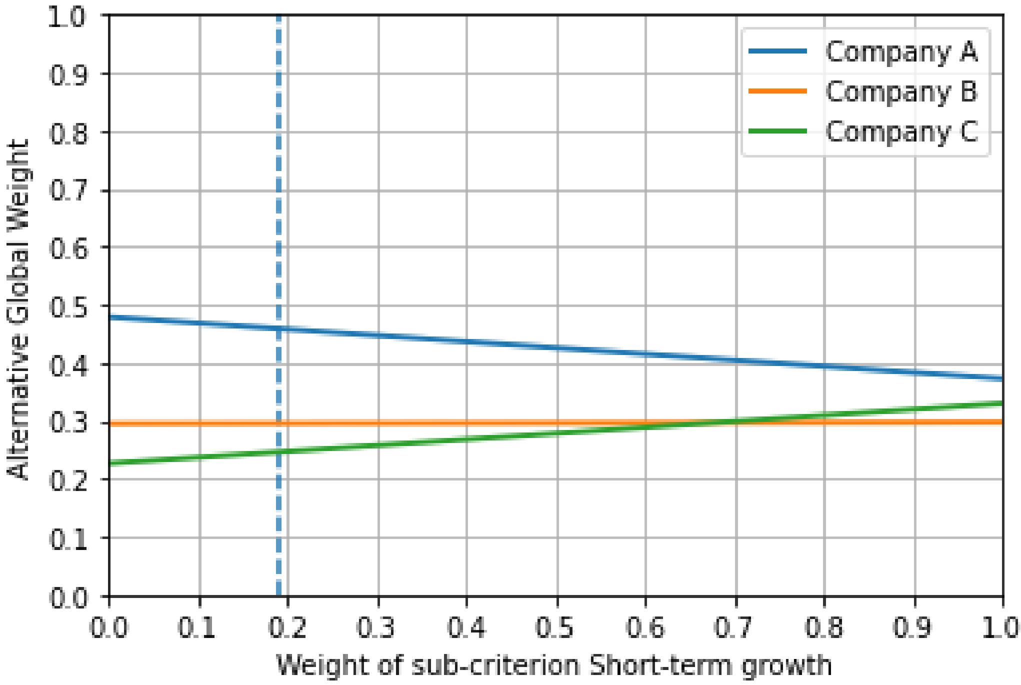
```
Sensitivity Analysis:

Rainbow Diagram for changing weight of main criterion Research
```



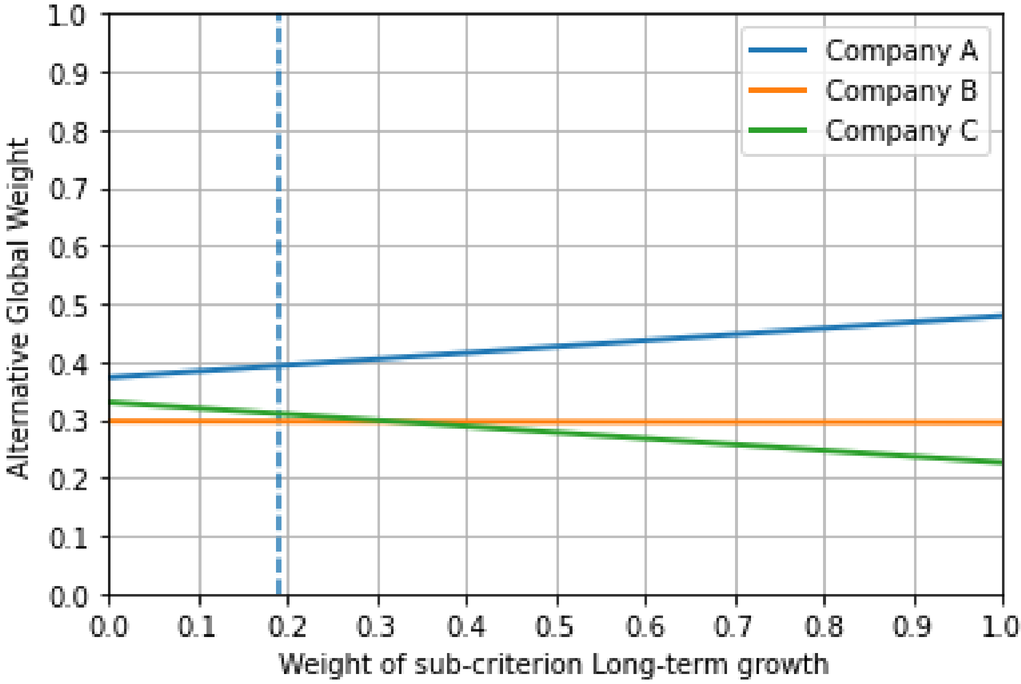
Rainbow Diagram for changing weight of main criterion Growth



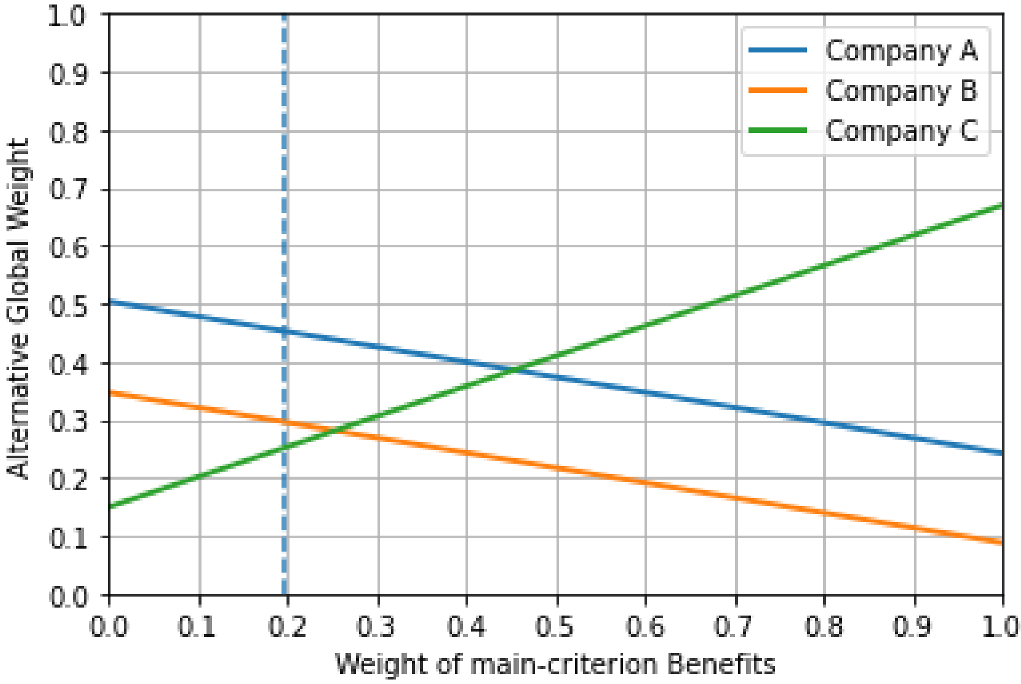
Rainbow Diagram for changing weight of sub-criterion Short-term growth



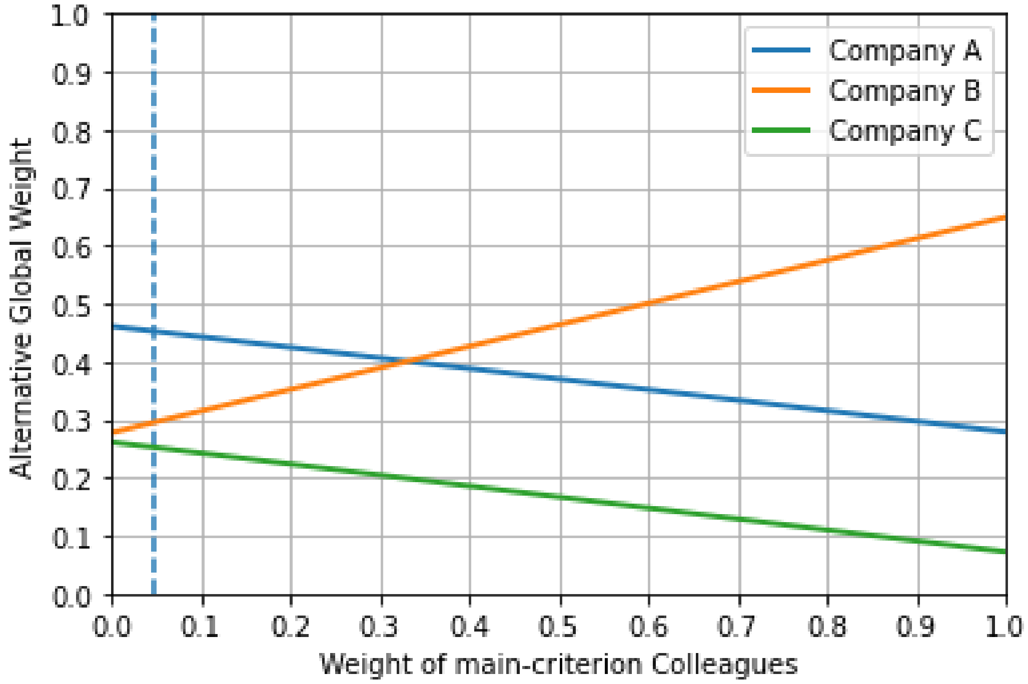
Rainbow Diagram for changing weight of sub-criterion Long-term growth



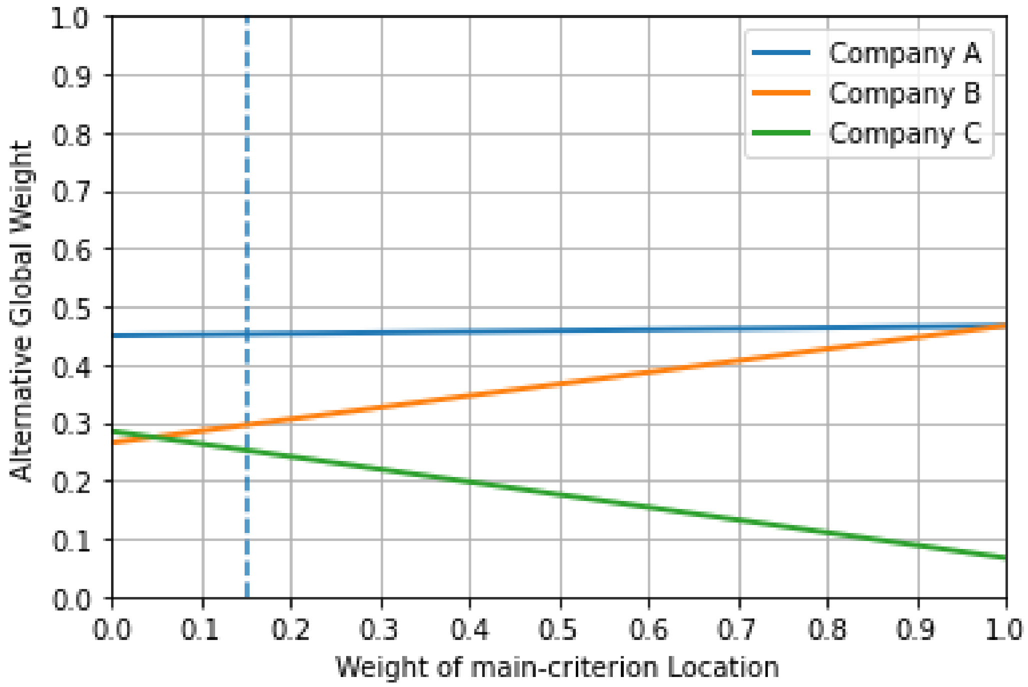
Rainbow Diagram for changing weight of main criterion Benefits



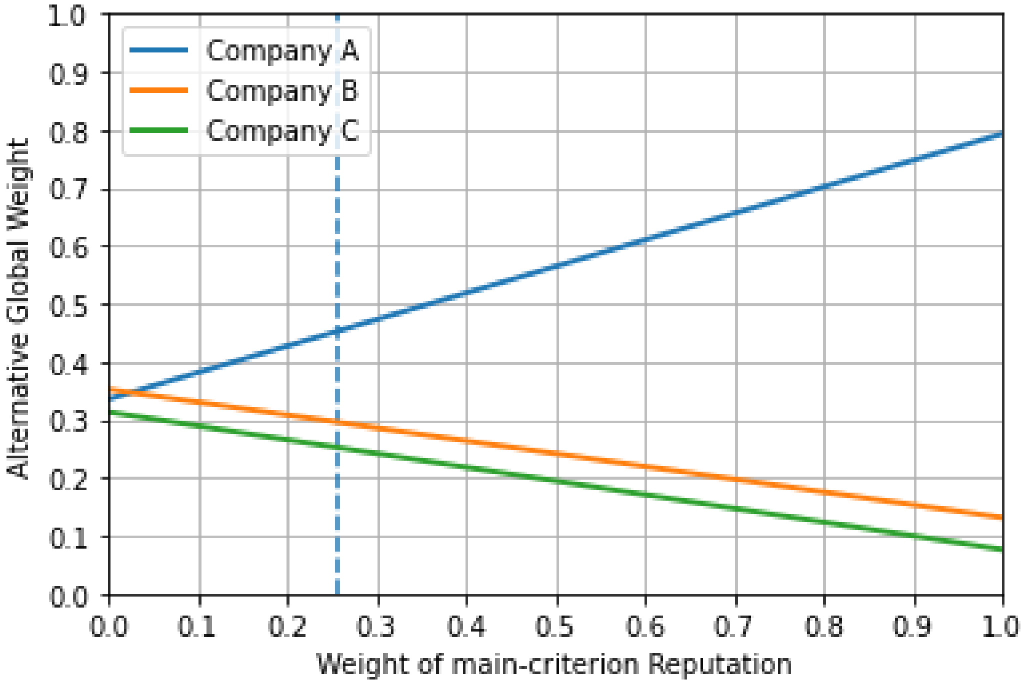
```
Rainbow Diagram for changing weight of main criterion Colleagues
```




```
Rainbow Diagram for changing weight of main criterion Location
```



```
Rainbow Diagram for changing weight of main criterion Reputation
```



10.3 Drug Counselling Center Relocation Problem (AHP model)

Source: 10.3_Center_Relocation_AHP_using_AHP4Lmodel_Class.ipynb

[1]:
```
""" Solve Drug Counseling Center Relocation Problem using AHP4Lmodel␣
    ↪Class """
from DecisionAnalysisPy import AHP4Lmodel
import numpy as np
```

[2]:
```
Goal = "Best Site for Relocation"
alternatives = ["Site 1", "Site 2", "Site 3", "Site 4", "Site 5", "Site␣
    ↪6"]

main_criteria = ["Good conditions for staff",
                 "Easy access for clients",
                 "Suitability of space for center's functions",
                 "Administrative convenience" ]

main_criteria_matrix = np.array([2, 2, 3,
                                    1, 2,
                                       1 ])
```

[3]:
```python
# Containers for data
sub_criteria = []
sub_criteria_matrices = []
alt_matrices = []

# Main Criterion 1: "Good Conditions for staff"
# List of subcriteria for Criterion 1
sub_criteria.append(["Office size",
                     "Convenience of staff commuting",
                     "Office attractiveness",
                     "Office privacy",
                     "Staff parking" ])

# Pairwise comparison of subcriteria for Criterion 1
sub_criteria_matrices.append(np.array([2, 3, 3, 3,
                                          2, 2, 2,
                                             1, 1,
                                                1 ]))

# Pairwise comparison of alternatives w.r.t. each sub-criterion
alt_matrices.append([np.array([2, 9, 2,  9,  2,
                                  5, 1,  5,  1,
                                    1/5, 1, 1/4,
                                         5,  1,
                                            1/4 ]),
                     np.array([2, 1/2, 5, 9,  2,
                                  1/3, 3, 6,  1,
                                       9, 9,  3,
                                          2,  1/3,
                                              1/6 ]),
                     np.array([1/3, 1/2, 3, 1/3, 1/3,
                                     1,  8,  1,   1,
                                         1,  1,   1,
                                             1/9, 1/8,
                                                   1 ]),
                     np.array([3, 2, 9,  3,   2,
                                  1, 3,  1,  1/2,
                                     4,  1,   1,
                                       1/3, 1/2,
                                              1 ]),

                     np.array([1/6, 1/3, 1, 1/9, 1/5,
                                     2,  6,       1/2,  1,
                                         3,  1/3, 1/2,
                                             1/9, 1/5,
                                                   2  ]) ])
```

[4]:
```python
# Main Criterion 2: "Easy access for clients"
# List of subcriteria for Criterion 2
sub_criteria.append(["Closeness to client's homes",
                     "Access to public transportation" ])

# Pairwise comparison of subcriteria for Criterion 2
sub_criteria_matrices.append(np.array([ 1 ]))

# Pairwise comparison of alternatives w.r.t. each sub-criterion
alt_matrices.append([np.array([1, 3, 1/2, 1/3,  1,
                                  3, 1/2, 1/3,  1,
                                     1/5, 1/9, 1/3,
                                          1/2,  2,
                                                3 ]),
                     np.array([1, 1, 1, 7,  1,
                                  1, 1, 7,  1,
                                     2, 9,  1,
                                        5,  1,
                                           1/7 ]) ])
```

[5]:
```python
# Main Criterion 3: "Suitability of space for for center's functions"
# List of subcriteria for Criterion 3
sub_criteria.append(["Number and suitability of counseling rooms",
                     "Number and suitability of conference rooms",
                     "Suitability of reception and waiting area" ])

# Pairwise comparison of subcriteria for Criterion 3
sub_criteria_matrices.append(np.array([ 2, 2,
                                           2 ]))

# Pairwise comparison of alternatives w.r.t. each sub-criterion
alt_matrices.append([np.array([1/8, 2, 1/5, 1/9, 1/5,
                                   9,  2,   1,   2,
                                      1/9, 1/9, 1/9,
                                           1/2,  1,
                                                 2 ]),
                    np.array([1, 6, 6, 1/2,  2,
                                 5,        5, 1/2,         2,
                                    1, 1/9, 1/3,
                                       1/9,        1/3,
                                             3 ]),
                    np.array([1, 1, 5,  2,   2,
                                 1,        4, 1/2,  1,
                                    5, 1/2,         2,
                                       1/9, 1/3,
                                             3  ])] )
```

[6]:
```
# Main Criterion 4: "Administrative Convenience"
# List of subcriteria for Criterion 4
sub_criteria.append(["Adequacy of space for admin work",
                     "Flexibility of the space layout" ])

# Pairwise comparison of subcriteria for Criterion 4
sub_criteria_matrices.append(np.array([ 2 ]))

# Pairwise comparison of alternatives w.r.t. each sub-criterion
alt_matrices.append([np.array([1/7, 1/5, 1/9, 1/5, 1/6,
                                     1,   1,   1,   1,
                                         1/2,  1,   1,
                                               2,   2,
                                                    1 ]),
                     np.array([1/4, 1/5, 1/9,  1,       1/4,
                                      1,  2,   4,    1,
                                         1/2,  5,    1,
                                               9,    1,
                                                    1/4 ]) ])

## End of Model Definition and Data ##
```

[7]:
```
# Create an instance of a 4-Level AHP model
DC_relocate = AHP4Lmodel(Goal, main_criteria, main_criteria_matrix,
                        sub_criteria, sub_criteria_matrices,
                        alternatives, alt_matrices)
```

[8]:
```
# Take a look a the model structure and data
DC_relocate.model()
```

```
Goal: Best Site for Relocation

Number of main criteria = 4
Main Criteria: ['Good conditions for staff', 'Easy access for clients',
"Suitability of space for center's functions", 'Administrative␣
 ↪convenience']

Pairwise comparison w.r.t. Goal Best Site for Relocation:
[[ 1    2    2    3  ]
 [1/2   1    1    2  ]
 [1/2   1    1    1  ]
 [1/3  1/2   1    1  ]]

Main Criteron 1: Good conditions for staff
  Number of sub-criteria = 5
  ['Office size', 'Convenience of staff commuting', 'Office␣
 ↪attractiveness',
```

```
'Office privacy', 'Staff parking']
Pairwise comparison w.r.t {self.MC[cr]}:
[[ 1    2    3    3    3  ]
 [1/2   1    2    2    2  ]
 [1/3  1/2   1    1    1  ]
 [1/3  1/2   1    1    1  ]
 [1/3  1/2   1    1    1  ]]

Number of alternatives = 6
['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site 6']

Pairwise comparison w.r.t.  Office size
[[ 1    2    9    2    9    2  ]
 [1/2   1    5    1    5    1  ]
 [1/9  1/5   1   1/5   1   1/4 ]
 [1/2   1    5    1    5    1  ]
 [1/9  1/5   1   1/5   1   1/4 ]
 [1/2   1    4    1    4    1  ]]

Number of alternatives = 6
['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site 6']

Pairwise comparison w.r.t.  Convenience of staff commuting
[[ 1    2   1/2   5    9    2  ]
 [1/2   1   1/3   3    6    1  ]
 [ 2    3    1    9    9    3  ]
 [1/5  1/3  1/9   1    2   1/3 ]
 [1/9  1/6  1/9  1/2   1   1/6 ]
 [1/2   1   1/3   3    6    1  ]]

Number of alternatives = 6
['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site 6']

Pairwise comparison w.r.t.  Office attractiveness
[[ 1   1/3  1/2   3   1/3  1/3 ]
 [ 3    1    1    8    1    1  ]
 [ 2    1    1    1    1    1  ]
 [1/3  1/8   1    1   1/9  1/8 ]
 [ 3    1    1    9    1    1  ]
 [ 3    1    1    8    1    1  ]]

Number of alternatives = 6
['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site 6']

Pairwise comparison w.r.t.  Office privacy
[[ 1    3    2    9    3    2  ]
 [1/3   1    1    3    1   1/2 ]
 [1/2   1    1    4    1    1  ]
```

```
 [1/9  1/3  1/4   1    1/3  1/2 ]
 [1/3   1    1    3     1    1  ]
 [1/2   2    1    2     1    1  ]]

Number of alternatives = 6
['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site 6']

Pairwise comparison w.r.t.  Staff parking
[[ 1   1/6  1/3   1    1/9  1/5 ]
 [ 6    1    2    6    1/2   1  ]
 [ 3   1/2   1    3    1/3  1/2 ]
 [ 1   1/6  1/3   1    1/9  1/5 ]
 [ 9    2    3    9     1    2  ]
 [ 5    1    2    5    1/2   1  ]]

Main Criteron 2: Easy access for clients
  Number of sub-criteria = 2
  ["Closeness to client's homes", 'Access to public transportation']
Pairwise comparison w.r.t {self.MC[cr]}:
[[ 1    1  ]
 [ 1    1  ]]

Number of alternatives = 6
['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site 6']

Pairwise comparison w.r.t.  Closeness to client's homes
[[ 1    1    3   1/2  1/3   1  ]
 [ 1    1    3   1/2  1/3   1  ]
 [1/3  1/3   1   1/5  1/9  1/3 ]
 [ 2    2    5    1   1/2   2  ]
 [ 3    3    9    2    1    3  ]
 [ 1    1    3   1/2  1/3   1  ]]

Number of alternatives = 6
['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site 6']

Pairwise comparison w.r.t.  Access to public transportation
[[ 1    1    1    1    7    1  ]
 [ 1    1    1    1    7    1  ]
 [ 1    1    1    2    9    1  ]
 [ 1    1   1/2   1    5    1  ]
 [1/7  1/7  1/9  1/5   1   1/7 ]
 [ 1    1    1    1    7    1  ]]

Main Criteron 3: Suitability of space for center's functions
  Number of sub-criteria = 3
  ['Number and suitability of counseling rooms', 'Number and suitability␣
 ↪of
```

```
conference rooms', 'Suitability of reception and waiting area']
Pairwise comparison w.r.t {self.MC[cr]}:
[[ 1    2    2  ]
 [1/2   1    2  ]
 [1/2  1/2   1  ]]

Number of alternatives = 6
['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site 6']

Pairwise comparison w.r.t.  Number and suitability of counseling rooms
[[ 1   1/8   2   1/5  1/9  1/5 ]
 [ 8    1    9    2    1    2  ]
 [1/2  1/9   1   1/9  1/9  1/9 ]
 [ 5   1/2   9    1   1/2   1  ]
 [ 9    1    9    2    1    2  ]
 [ 5   1/2   9    1   1/2   1  ]]

Number of alternatives = 6
['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site 6']

Pairwise comparison w.r.t.  Number and suitability of conference rooms
[[ 1    1    6    6   1/2   2  ]
 [ 1    1    5    5   1/2   2  ]
 [1/6  1/5   1    1   1/9  1/3 ]
 [1/6  1/5   1    1   1/9  1/3 ]
 [ 2    2    9    9    1    3  ]
 [1/2  1/2   3    3   1/3   1  ]]

Number of alternatives = 6
['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site 6']

Pairwise comparison w.r.t.  Suitability of reception and waiting area
[[ 1    1    1    5    2    2  ]
 [ 1    1    1    4   1/2   1  ]
 [ 1    1    1    5   1/2   2  ]
 [1/5  1/4  1/5   1   1/9  1/3 ]
 [1/2   2    2    9    1    3  ]
 [1/2   1   1/2   3   1/3   1  ]]

Main Criteron 4: Administrative convenience
  Number of sub-criteria = 2
  ['Adequacy of space for admin work', 'Flexibility of the space layout']
Pairwise comparison w.r.t {self.MC[cr]}:
[[ 1    2  ]
 [1/2   1  ]]

Number of alternatives = 6
['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site 6']
```

```
Pairwise comparison w.r.t.  Adequacy of space for admin work
[[ 1   1/7  1/5  1/9  1/5  1/6 ]
 [ 7    1    1    1    1    1  ]
 [ 5    1    1   1/2   1    1  ]
 [ 9    1    2    1    2    2  ]
 [ 5    1    1   1/2   1    1  ]
 [ 6    1    1   1/2   1    1  ]]

Number of alternatives = 6
['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site 6']

Pairwise comparison w.r.t.  Flexibility of the space layout
[[ 1   1/4  1/5  1/9   1   1/4 ]
 [ 4    1    1    2    4    1  ]
 [ 5    1    1   1/2   5    1  ]
 [ 9   1/2   2    1    9    1  ]
 [ 1   1/4  1/5  1/9   1   1/4 ]
 [ 4    1    1    1    4    1  ]]
```

[9]:
```python
# Solve the model
DC_relocate.solve(method="Algebra")
```

```
Goal: Best Site for Relocation
Alternatives: ['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site
 ↪6']
Main Criteria: ['Good conditions for staff', 'Easy access for clients',
"Suitability of space for center's functions", 'Administrative
 ↪convenience']

Pairwise comparison of main criteria w.r.t. Goal Best Site for
 ↪Relocation:
[[ 1    2    2    3  ]
 [1/2   1    1    2  ]
 [1/2   1    1    1  ]
 [1/3  1/2   1    1  ]]
Lambda = 4.045819, CI= 0.015273, CR= 0.016970
Main criteria weights= [0.425784 0.231236 0.194548 0.148431]
Pairwise comparison of sub-criteria for Good conditions for staff:
[[ 1    2    3    3    3  ]
 [1/2   1    2    2    2  ]
 [1/3  1/2   1    1    1  ]
 [1/3  1/2   1    1    1  ]
 [1/3  1/2   1    1    1  ]]
Lambda = 5.009960, CI= 0.002490, CR= 0.002223
Sub-citeria weights= [0.394537 0.23431  0.123718 0.123718 0.123718]
[[ 1    2    9    2    9    2  ]
```

```
 [1/2   1    5    1    5    1  ]
 [1/9  1/5   1   1/5   1   1/4 ]
 [1/2   1    5    1    5    1  ]
 [1/9  1/5   1   1/5   1   1/4 ]
 [1/2   1    4    1    4    1  ]]
Lambda = 6.007543, CI= 0.001509, CR= 0.001217
Alternative weights= [0.365332 0.189325 0.040006 0.189325 0.040006 0.
 ↪176007]
[[ 1    2   1/2   5    9    2  ]
 [1/2   1   1/3   3    6    1  ]
 [ 2    3    1    9    9    3  ]
 [1/5  1/3  1/9   1    2   1/3 ]
 [1/9  1/6  1/9  1/2   1   1/6 ]
 [1/2   1   1/3   3    6    1  ]]
Lambda = 6.066245, CI= 0.013249, CR= 0.010685
Alternative weights= [0.246725 0.139716 0.397855 0.047928 0.028059 0.
 ↪139716]
[[ 1   1/3  1/2   3   1/3  1/3 ]
 [ 3    1    1    8    1    1  ]
 [ 2    1    1    1    1    1  ]
 [1/3  1/8   1    1   1/9  1/8 ]
 [ 3    1    1    9    1    1  ]
 [ 3    1    1    8    1    1  ]]
Lambda = 6.561165, CI= 0.112233, CR= 0.090510
Alternative weights= [0.08379  0.231178 0.165183 0.049889 0.238782 0.
 ↪231178]
[[ 1    3    2    9    3    2  ]
 [1/3   1    1    3    1   1/2 ]
 [1/2   1    1    4    1    1  ]
 [1/9  1/3  1/4   1   1/3  1/2 ]
 [1/3   1    1    3    1    1  ]
 [1/2   2    1    2    1    1  ]]
Lambda = 6.120013, CI= 0.024003, CR= 0.019357
Alternative weights= [0.36614  0.126205 0.157383 0.048753 0.139446 0.
 ↪162073]
[[ 1   1/6  1/3   1   1/9  1/5 ]
 [ 6    1    2    6   1/2   1  ]
 [ 3   1/2   1    3   1/3  1/2 ]
 [ 1   1/6  1/3   1   1/9  1/5 ]
 [ 9    2    3    9    1    2  ]
 [ 5    1    2    5   1/2   1  ]]
Lambda = 6.013523, CI= 0.002705, CR= 0.002181
Alternative weights= [0.039471 0.219426 0.114984 0.039471 0.380349 0.
 ↪206299]
Pairwise comparison of sub-criteria for Easy access for clients:
[[ 1    1  ]
 [ 1    1  ]]
Lambda = 2.000000, CI= 0.000000, CR= 0.000000
```

```
Sub-citeria weights= [0.5 0.5]
[[ 1    1    3   1/2  1/3   1  ]
 [ 1    1    3   1/2  1/3   1  ]
 [1/3  1/3   1   1/5  1/9  1/3 ]
 [ 2    2    5    1   1/2   2  ]
 [ 3    3    9    2    1    3  ]
 [ 1    1    3   1/2  1/3   1  ]]
Lambda = 6.009998, CI= 0.002000, CR= 0.001613
Alternative weights= [0.119713 0.119713 0.041136 0.222109 0.377617 0.
 ↪119713]
[[ 1    1    1    1    7    1  ]
 [ 1    1    1    1    7    1  ]
 [ 1    1    1    2    9    1  ]
 [ 1    1   1/2   1    5    1  ]
 [1/7  1/7  1/9  1/5   1   1/7 ]
 [ 1    1    1    1    7    1  ]]
Lambda = 6.055665, CI= 0.011133, CR= 0.008978
Alternative weights= [0.192769 0.192769 0.229167 0.164636 0.027891 0.
 ↪192769]
Pairwise comparison of sub-criteria for Suitability of space for center's
functions:
[[ 1    2    2  ]
 [1/2   1    2  ]
 [1/2  1/2   1  ]]
Lambda = 3.053622, CI= 0.026811, CR= 0.046225
Sub-citeria weights= [0.493386 0.310814 0.1958  ]
[[ 1   1/8   2   1/5  1/9  1/5 ]
 [ 8    1    9    2    1    2  ]
 [1/2  1/9   1   1/9  1/9  1/9 ]
 [ 5   1/2   9    1   1/2   1  ]
 [ 9    1    9    2    1    2  ]
 [ 5   1/2   9    1   1/2   1  ]]
Lambda = 6.081230, CI= 0.016246, CR= 0.013102
Alternative weights= [0.036847 0.295021 0.024159 0.171447 0.30108  0.
 ↪171447]
[[ 1    1    6    6   1/2   2  ]
 [ 1    1    5    5   1/2   2  ]
 [1/6  1/5   1    1   1/9  1/3 ]
 [1/6  1/5   1    1   1/9  1/3 ]
 [ 2    2    9    9    1    3  ]
 [1/2  1/2   3    3   1/3   1  ]]
Lambda = 6.013523, CI= 0.002705, CR= 0.002181
Alternative weights= [0.219426 0.206299 0.039471 0.039471 0.380349 0.
 ↪114984]
[[ 1    1    1    5    2    2  ]
 [ 1    1    1    4   1/2   1  ]
 [ 1    1    1    5   1/2   2  ]
 [1/5  1/4  1/5   1   1/9  1/3 ]
```

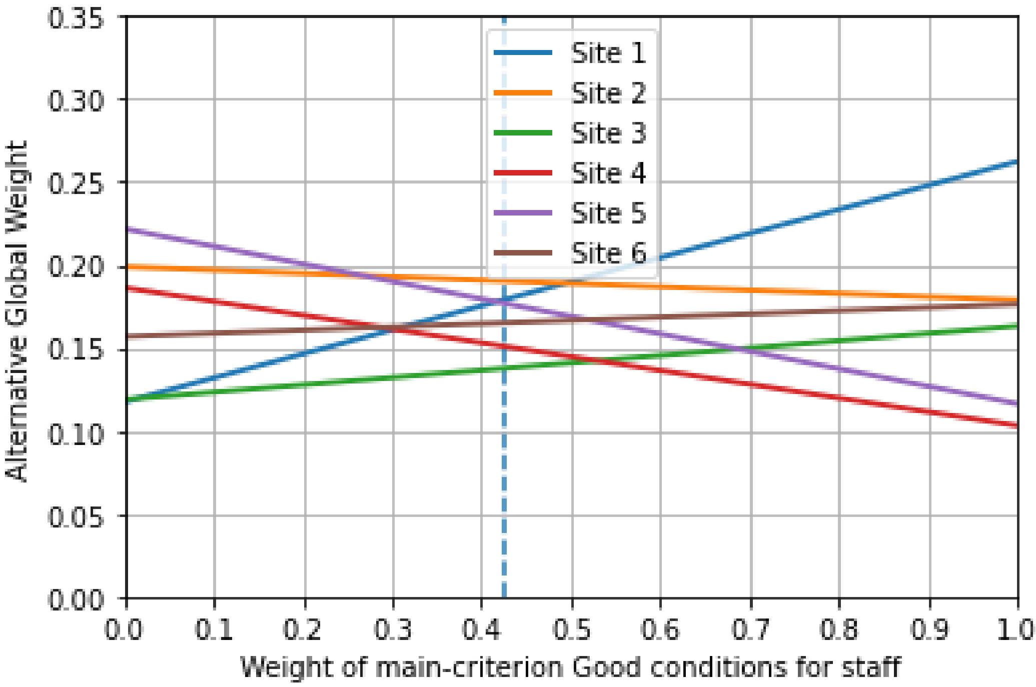
```
 [1/2   2    2    9    1    3  ]
 [1/2   1   1/2   3   1/3   1  ]]
Lambda = 6.240366, CI= 0.048073, CR= 0.038769
Alternative weights= [0.245092 0.155641 0.178851 0.036349 0.275579 0.
 ↪108488]
Pairwise comparison of sub-criteria for Administrative convenience:
[[ 1    2  ]
 [1/2   1  ]]
Lambda = 2.000000, CI= 0.000000, CR= 0.000000
Sub-citeria weights= [0.666667 0.333333]
[[ 1   1/7  1/5  1/9  1/5  1/6 ]
 [ 7    1    1    1    1    1  ]
 [ 5    1    1   1/2   1    1  ]
 [ 9    1    2    1    2    2  ]
 [ 5    1    1   1/2   1    1  ]
 [ 6    1    1   1/2   1    1  ]]
Lambda = 6.054173, CI= 0.010835, CR= 0.008738
Alternative weights= [0.030055 0.194962 0.161437 0.285707 0.161437 0.
 ↪166402]
[[ 1   1/4  1/5  1/9   1   1/4 ]
 [ 4    1    1    2    4    1  ]
 [ 5    1    1   1/2   5    1  ]
 [ 9   1/2   2    1    9    1  ]
 [ 1   1/4  1/5  1/9   1   1/4 ]
 [ 4    1    1    1    4    1  ]]
Lambda = 6.263322, CI= 0.052664, CR= 0.042471
Alternative weights= [0.042343 0.244746 0.191482 0.278862 0.042343 0.
 ↪200223]

Results:
  Site 1 : 0.179105
  Site 2 : 0.190379
  Site 3 : 0.137681
  Site 4 : 0.150873
  Site 5 : 0.176830
  Site 6 : 0.165132

Sorted Results:
  Site 2 : 0.190379
  Site 1 : 0.179105
  Site 5 : 0.176830
  Site 6 : 0.165132
  Site 4 : 0.150873
  Site 3 : 0.137681
```

[9]:
```
{'Site 1': 0.17910508959000196,
 'Site 2': 0.1903794996140339,
 'Site 3': 0.13768094135344633,
```

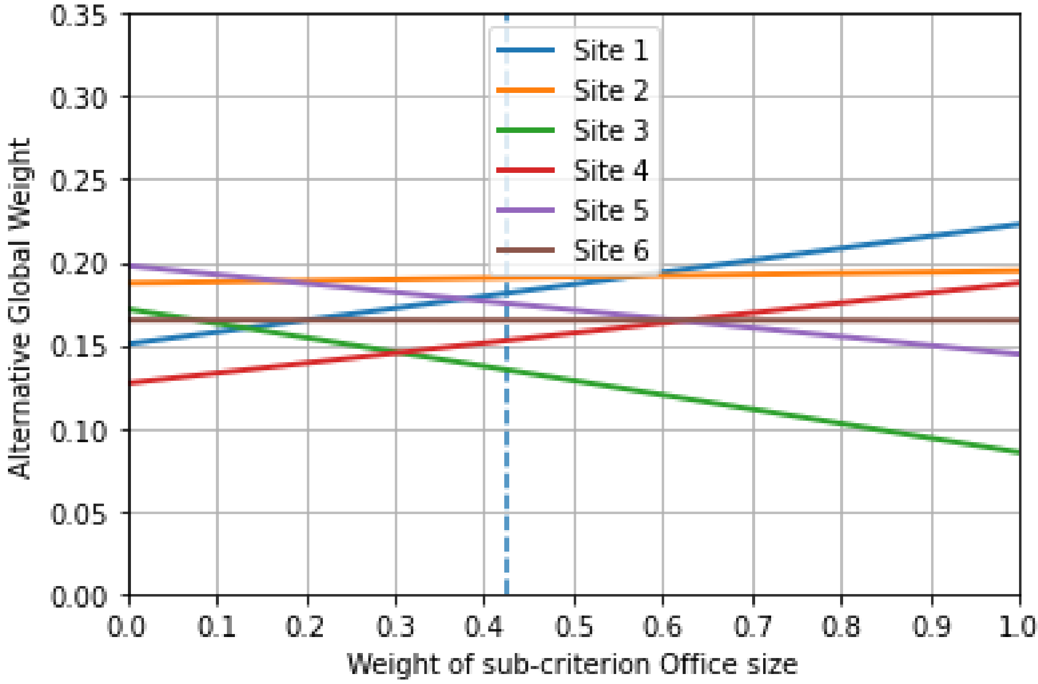
```
 'Site 4': 0.1508731589835098,
 'Site 5': 0.1768296957923447,
 'Site 6': 0.1651316146666633}
```

[10]:
```
# Do sensitivity analysis
DC_relocate.sensit(ymax=0.35, ystep=0.05)
```

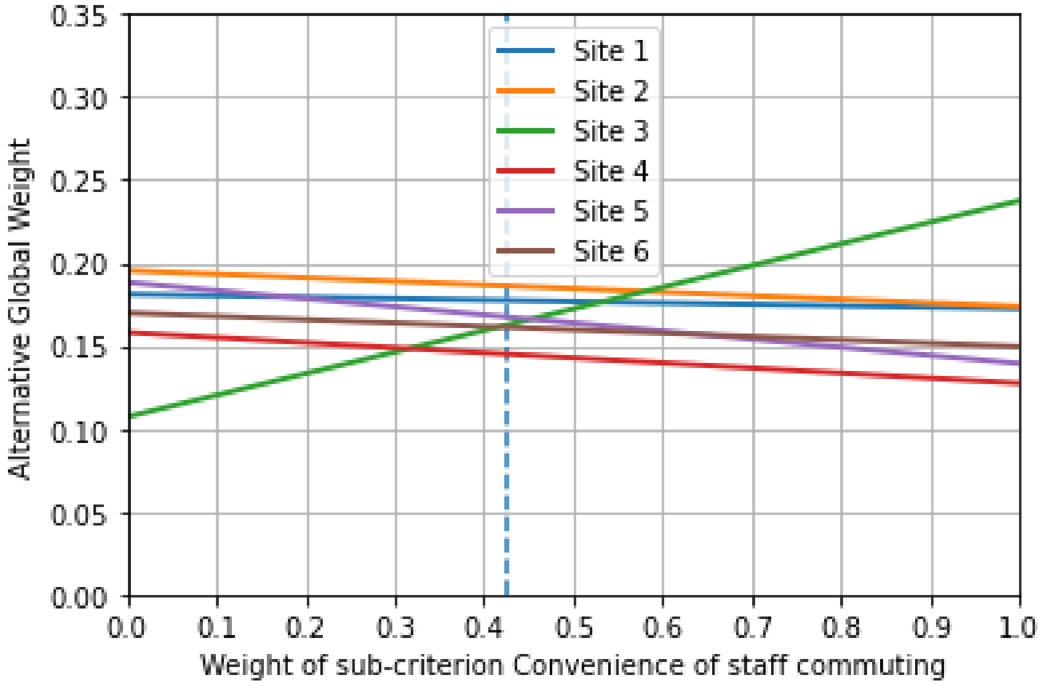
```
Sensitivity Analysis:

Rainbow Diagram for changing weight of main criterion Good conditions␣
 ↪for staff
```



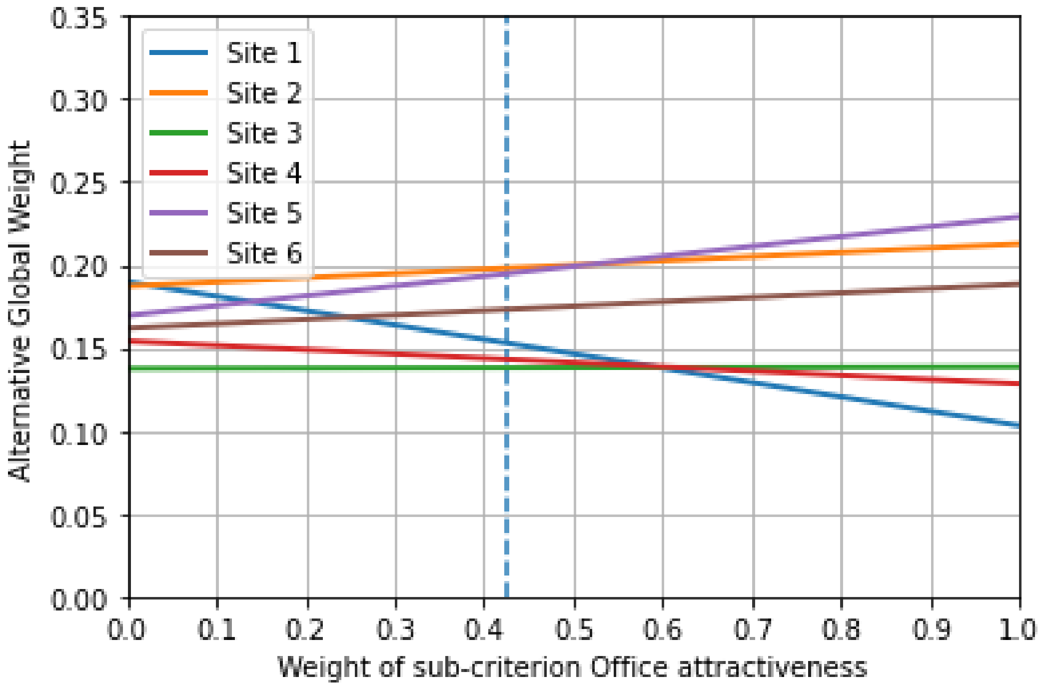

```
Rainbow Diagram for changing weight of sub-criterion Office size
```



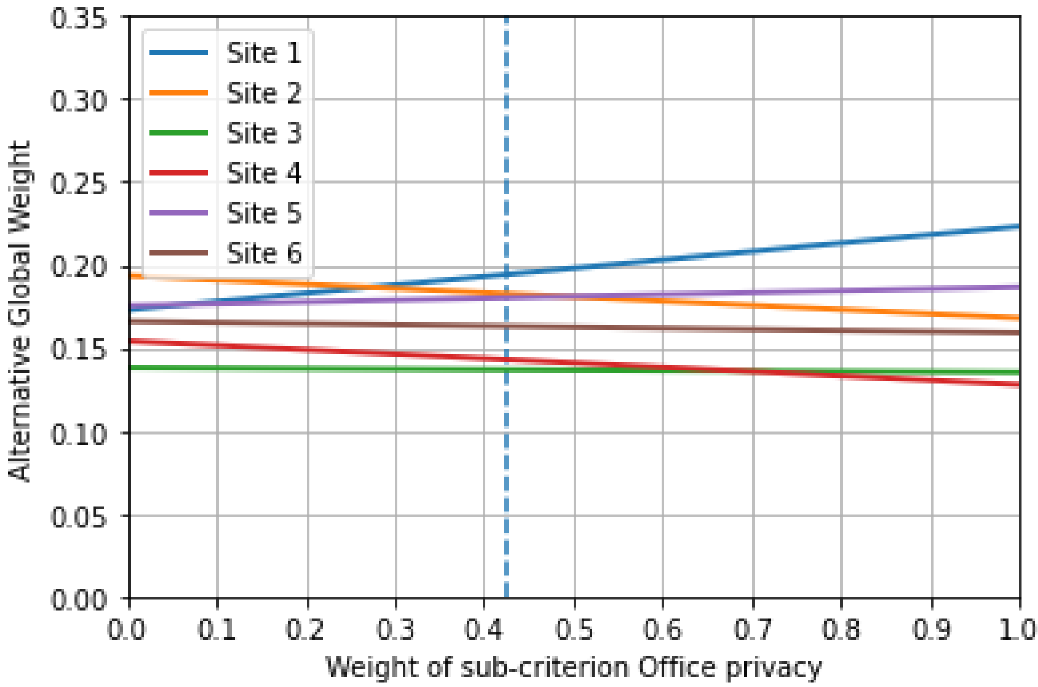
Rainbow Diagram for changing weight of sub-criterion Convenience of staff commuting



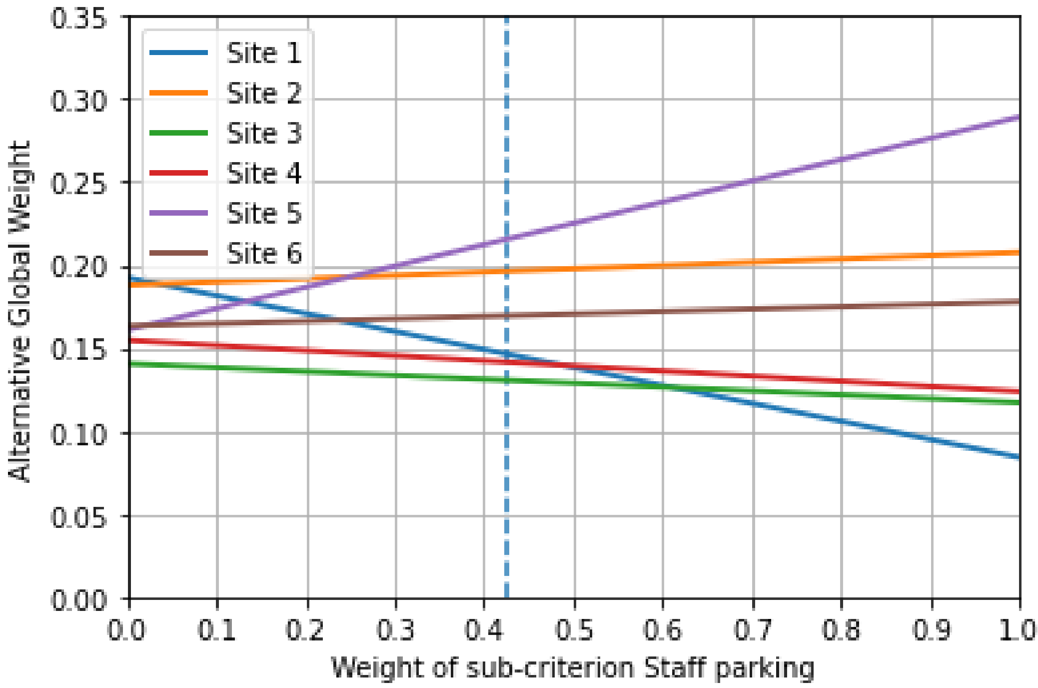
```
Rainbow Diagram for changing weight of sub-criterion Office␣
↪attractiveness
```



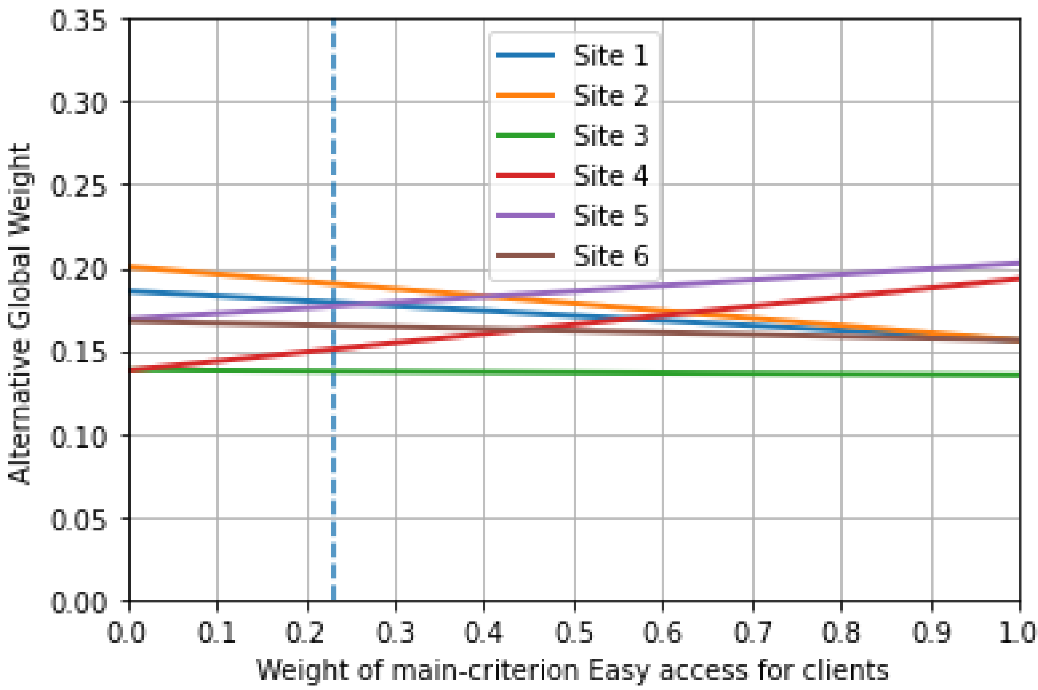

```
Rainbow Diagram for changing weight of sub-criterion Office privacy
```



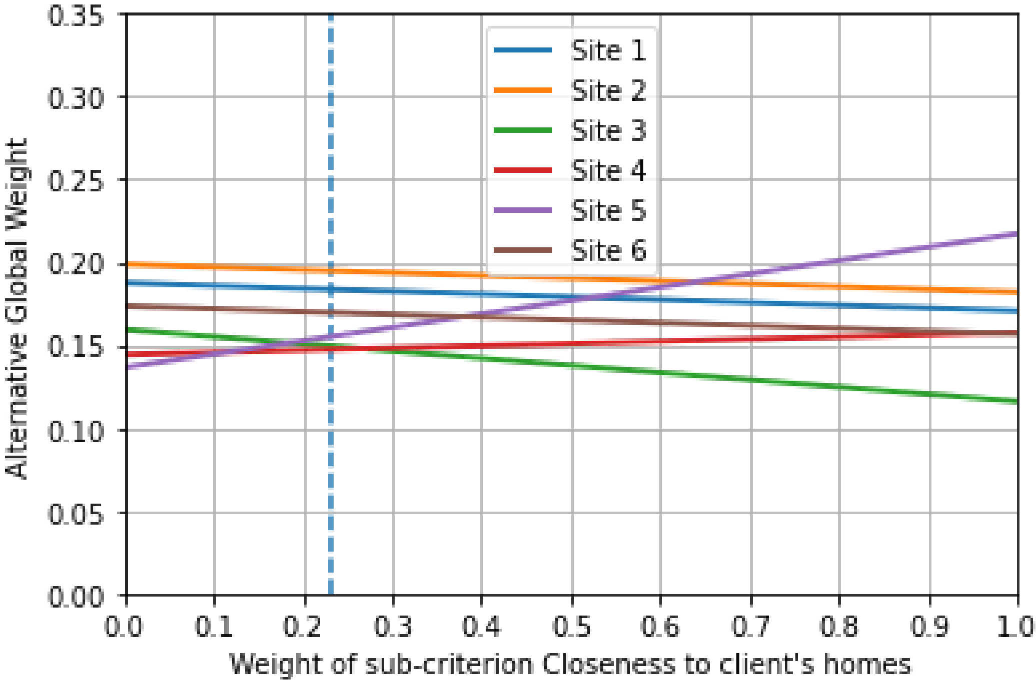
```
Rainbow Diagram for changing weight of sub-criterion Staff parking
```



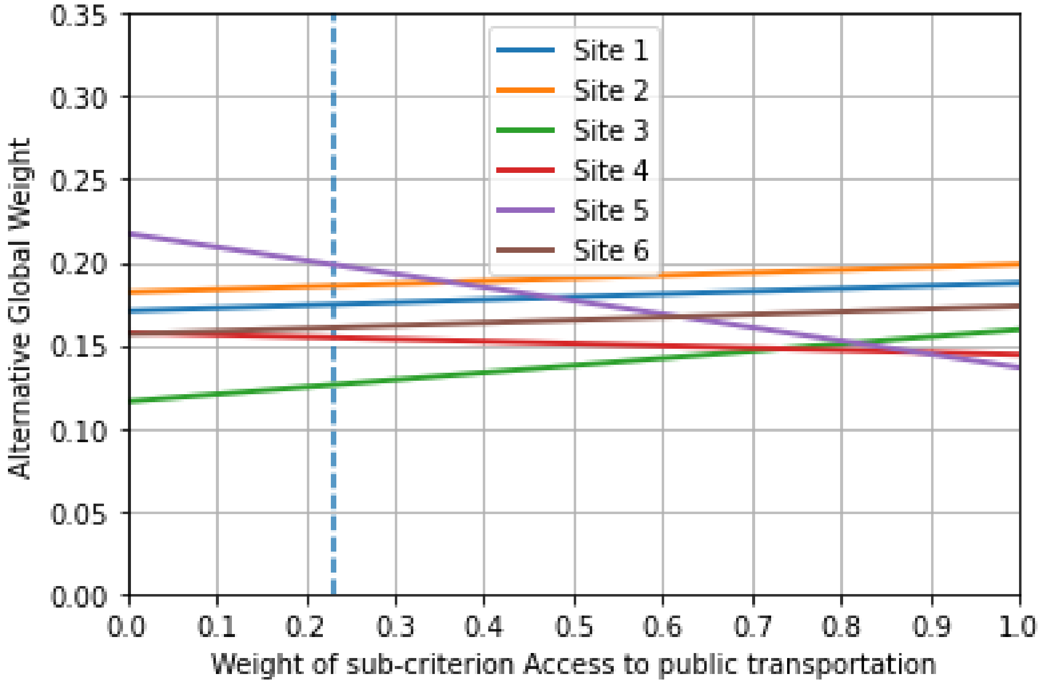

```
Rainbow Diagram for changing weight of main criterion Easy access for␣
↪clients
```



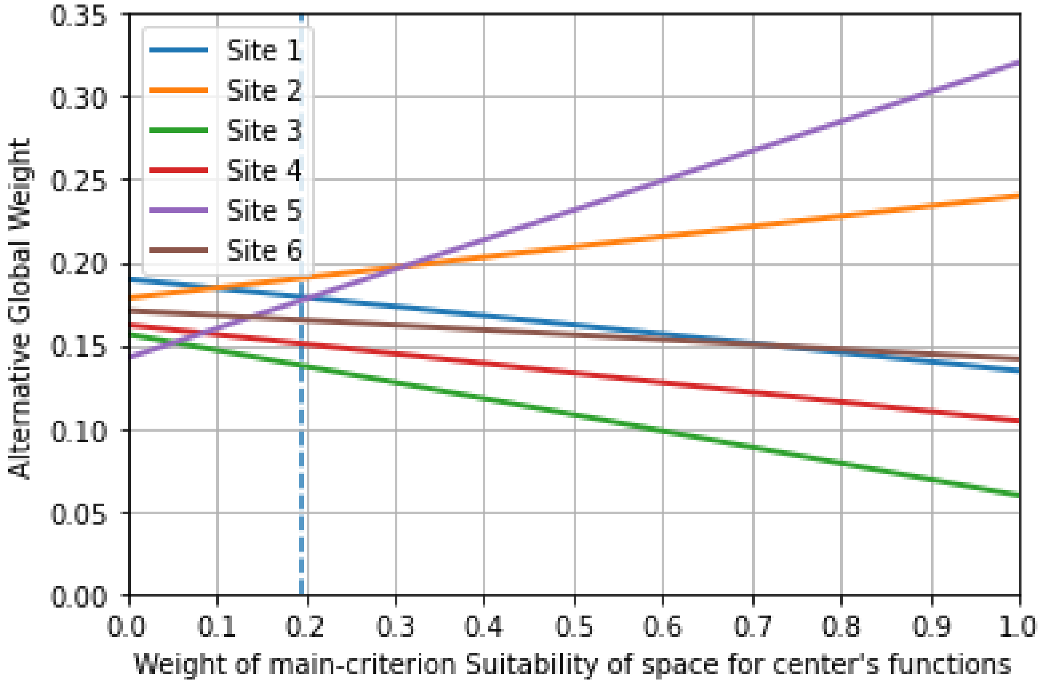
```
Rainbow Diagram for changing weight of sub-criterion Closeness to␣
↪client's homes
```



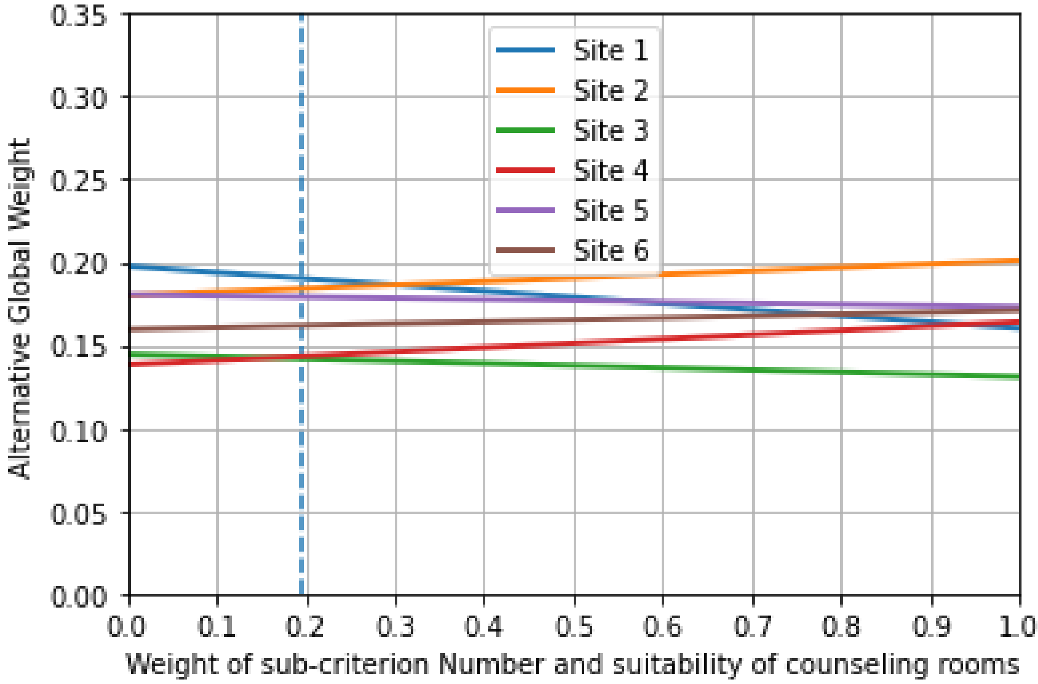

```
Rainbow Diagram for changing weight of sub-criterion Access to public
transportation
```



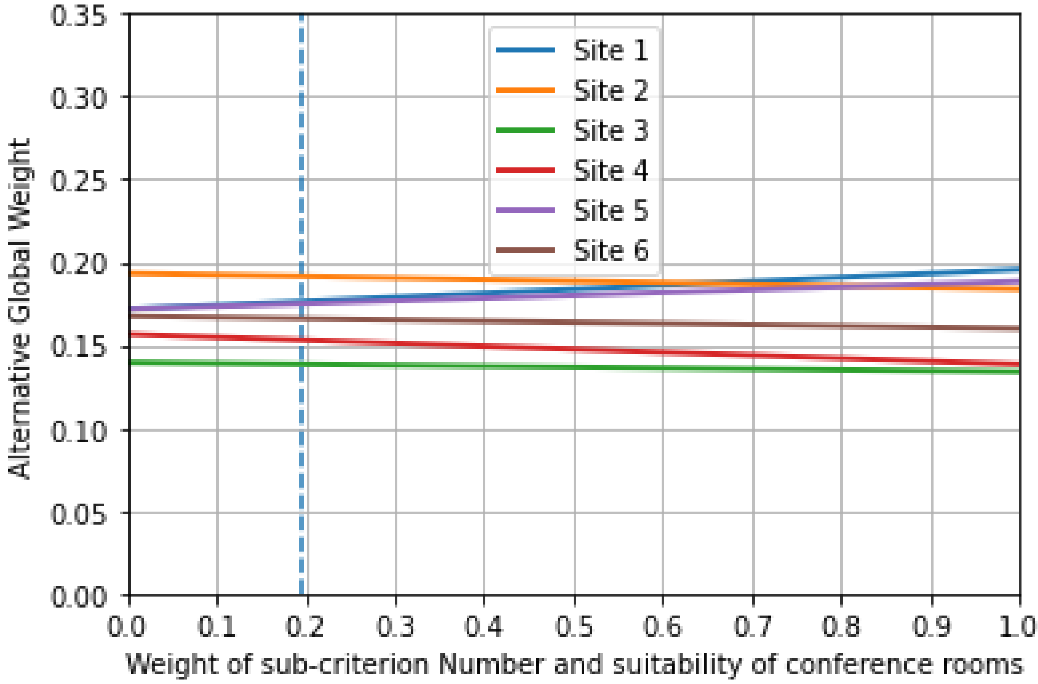

```
Rainbow Diagram for changing weight of main criterion Suitability of␣
↪space for
center's functions
```



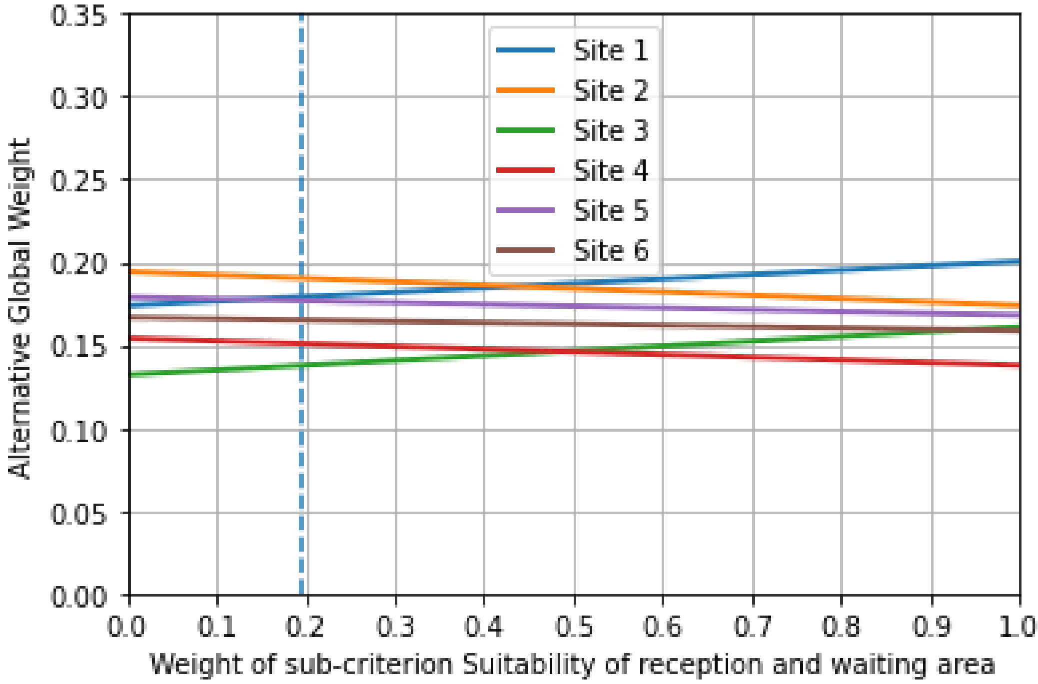
```
Rainbow Diagram for changing weight of sub-criterion Number and␣
↪suitability of
counseling rooms
```



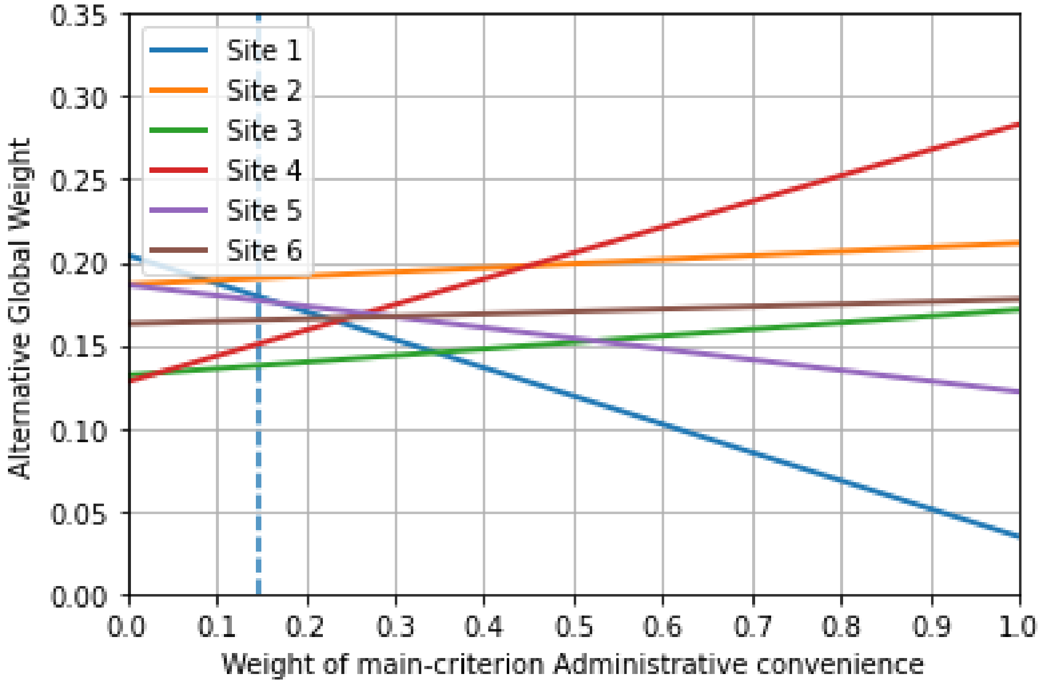

```
Rainbow Diagram for changing weight of sub-criterion Number and␣
↪suitability of
conference rooms
```



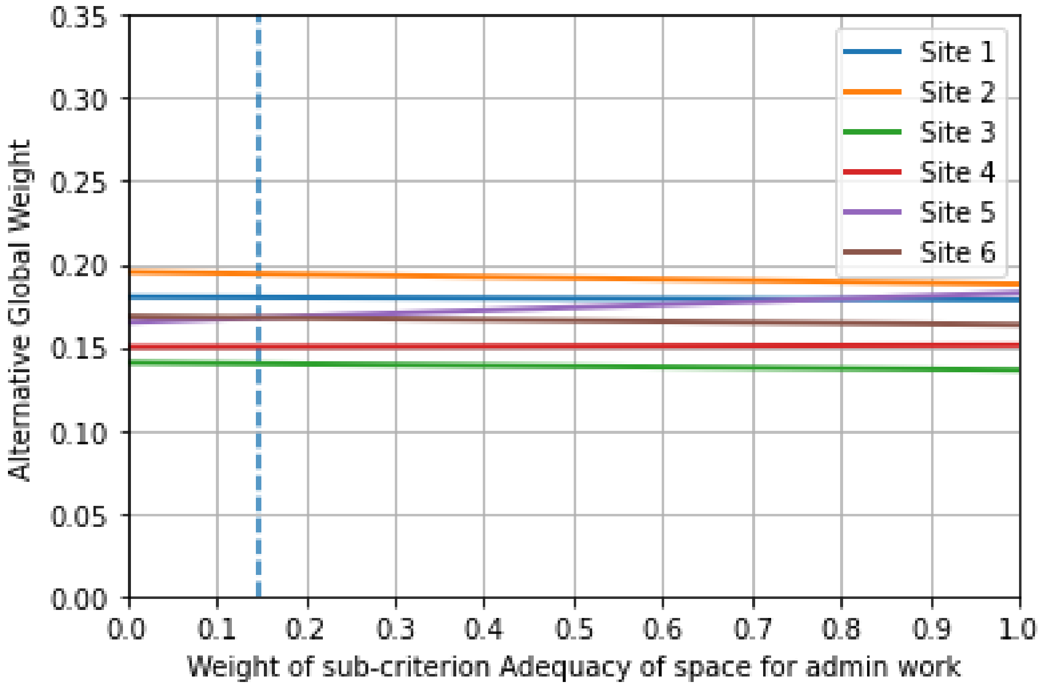
Rainbow Diagram for changing weight of sub-criterion Suitability of
↪reception
and waiting area



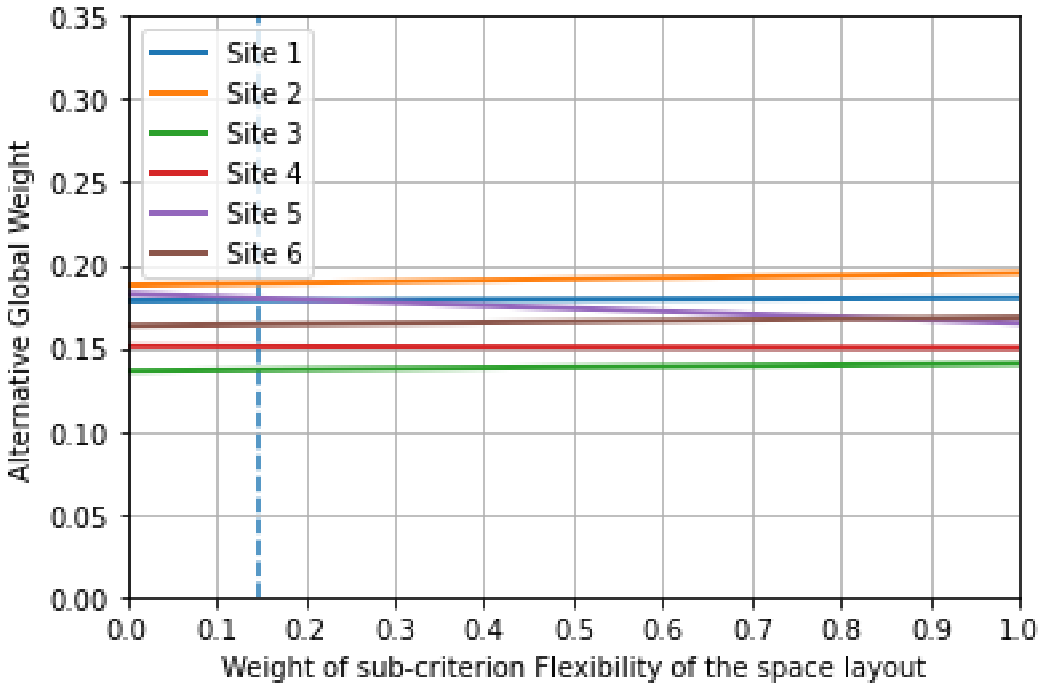
```
Rainbow Diagram for changing weight of main criterion Administrative␣
↪convenience
```




```
Rainbow Diagram for changing weight of sub-criterion Adequacy of space␣
↪for admin
work
```




```
Rainbow Diagram for changing weight of sub-criterion Flexibility of the␣
  ↪space
layout
```



10.4 Job Selection Problem with Sensitivity Analysis

Source: 9.5_Solve_Job_Selection_Problem_using_AHP4Lmodel_Class.ipynb

[1]:
```python
""" Solve Job Selection Problem and perform Sensitivity Analsyis
    using AHP4Lmodel Class. This example shows how to input a 3-Level
    AHP model using AHP4Lmodel Class instead of AHP3Lmodel Class """
from DecisionAnalysisPy import AHP4Lmodel
import numpy as np
```

[2]:
```python
Goal = "Job Satisfaction"
alternatives = ["Company A", "Company B", "Company C"]

main_criteria = ["Research",   "Growth",   "Benefits",
                 "Colleagues", "Location", "Reputation"]

main_criteria_matrix = np.array([1, 1, 4,  1,  1/2,
                                    2, 4,  1,  1/2,
                                       5,  3,  1/2,
                                          1/3, 1/3,
                                                1  ])
```

[3]:
```python
# Containers for model data
sub_criteria = []
sub_criteria_matrices = []
alt_matrices = []

# Main Criterion 1: "Research"
# List of subcriteria for Criterion 1
sub_criteria.append(None)
# Pairwise comparison of subcriteria for Criterion 1
sub_criteria_matrices.append(None)
# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_matrices.append([np.array([1/4, 1/2, 3 ])])
```

[4]:
```python
# Main Criterion 2: "Growth"
# List of subcriteria for Criterion 2
sub_criteria.append(None)
# Pairwise comparison of subcriteria for Criterion 2
sub_criteria_matrices.append(None)
# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_matrices.append([ np.array([1/4, 1/5, 1/2 ])])
```

[5]:
```python
# Main Criterion 3: "Benefits"
# List of subcriteria for Criterion 3
sub_criteria.append(None)
# Pairwise comparison of subcriteria for Criterion 3
sub_criteria_matrices.append(None)
# Pairwise comparison of alternatives w.r.t. each subcriterion
```

```
alt_matrices.append([np.array([3, 1/3, 1/7 ])])
```

[6]:
```
# Main Criterion 4: "Colleagues"
# List of subcriteria for Criterion 4
sub_criteria.append(None)
# Pairwise comparison of subcriteria for Criterion 4
sub_criteria_matrices.append(None)
# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_matrices.append([np.array([1/3, 5, 7 ])])
```

[7]:
```
# Main Criterion 5: "Location"
# List of subcriteria for Criterion 5
sub_criteria.append(None)
# Pairwise comparison of subcriteria for Criterion 5
sub_criteria_matrices.append(None)
# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_matrices.append([np.array([1, 7, 7 ])])
```

[8]:
```
# Main Criterion 6: "Reputation"
# List of subcriteria for Criterion 6
sub_criteria.append(None)
# Pairwise comparison of subcriteria for Criterion 5
sub_criteria_matrices.append(None)
# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_matrices.append([np.array([7, 9, 2 ])])

# End of model definition and data
```

[9]:
```
# Create a 4-Level AHP model
JobSelect = AHP4Lmodel(Goal, main_criteria, main_criteria_matrix,
                       sub_criteria, sub_criteria_matrices,
                       alternatives, alt_matrices)
```

[10]:
```
# Get model structure and data
JobSelect.model()
```

```
Goal: Job Satisfaction

Number of main criteria = 6
Main Criteria: ['Research', 'Growth', 'Benefits', 'Colleagues',␣
 ↪'Location',
'Reputation']

Pairwise comparison w.r.t. Goal Job Satisfaction:
[[ 1    1    1    4    1    1/2 ]
 [ 1    1    2    4    1    1/2 ]
 [ 1   1/2   1    5    3    1/2 ]
```

```
 [1/4  1/4  1/5   1   1/3  1/3 ]
 [ 1    1   1/3   3    1    1  ]
 [ 2    2    2    3    1    1  ]]

Main Criteron 1: Research
Criterion Research has no sub-criterion

Number of alternatives = 3
['Company A', 'Company B', 'Company C']
Pairwise comparison w.r.t. Research
[[ 1   1/4  1/2 ]
 [ 4    1    3  ]
 [ 2   1/3   1  ]]

Main Criteron 2: Growth
Criterion Growth has no sub-criterion

Number of alternatives = 3
['Company A', 'Company B', 'Company C']
Pairwise comparison w.r.t. Growth
[[ 1   1/4  1/5 ]
 [ 4    1   1/2 ]
 [ 5    2    1  ]]

Main Criteron 3: Benefits
Criterion Benefits has no sub-criterion

Number of alternatives = 3
['Company A', 'Company B', 'Company C']
Pairwise comparison w.r.t. Benefits
[[ 1    3   1/3 ]
 [1/3   1   1/7 ]
 [ 3    7    1  ]]

Main Criteron 4: Colleagues
Criterion Colleagues has no sub-criterion

Number of alternatives = 3
['Company A', 'Company B', 'Company C']
Pairwise comparison w.r.t. Colleagues
[[ 1   1/3   5  ]
 [ 3    1    7  ]
 [1/5  1/7   1  ]]

Main Criteron 5: Location
Criterion Location has no sub-criterion

Number of alternatives = 3
```

```
['Company A', 'Company B', 'Company C']
Pairwise comparison w.r.t. Location
[[ 1    1    7  ]
 [ 1    1    7  ]
 [1/7  1/7   1  ]]

Main Criteron 6: Reputation
Criterion Reputation has no sub-criterion

Number of alternatives = 3
['Company A', 'Company B', 'Company C']
Pairwise comparison w.r.t. Reputation
[[ 1    7    9  ]
 [1/7   1    2  ]
 [1/9  1/2   1  ]]
```

[11]:
```
# Solve the model
result = JobSelect.solve(method="Algebra")
```

```
Goal: Job Satisfaction
Alternatives: ['Company A', 'Company B', 'Company C']
Main Criteria: ['Research', 'Growth', 'Benefits', 'Colleagues', 
 ↪'Location',
'Reputation']

Pairwise comparison of main criteria w.r.t. Goal Job Satisfaction:
[[ 1    1    1    4    1    1/2 ]
 [ 1    1    2    4    1    1/2 ]
 [ 1   1/2   1    5    3    1/2 ]
 [1/4  1/4  1/5   1   1/3   1/3 ]
 [ 1    1   1/3   3    1     1  ]
 [ 2    2    2    3    1     1  ]]
Lambda = 6.420344, CI= 0.084069, CR= 0.067797
Main criteria weights= [0.158408 0.189247 0.197997 0.04831  0.150245 0.
 ↪255792]
Inside None
[[ 1   1/4  1/2 ]
 [ 4    1    3  ]
 [ 2   1/3   1  ]]
Lambda = 3.018295, CI= 0.009147, CR= 0.015771
Alternative weights= [0.1365   0.625013 0.238487]
Inside None
[[ 1   1/4  1/5 ]
 [ 4    1   1/2 ]
 [ 5    2    1  ]]
Lambda = 3.024595, CI= 0.012298, CR= 0.021203
Alternative weights= [0.09739  0.333069 0.569541]
```

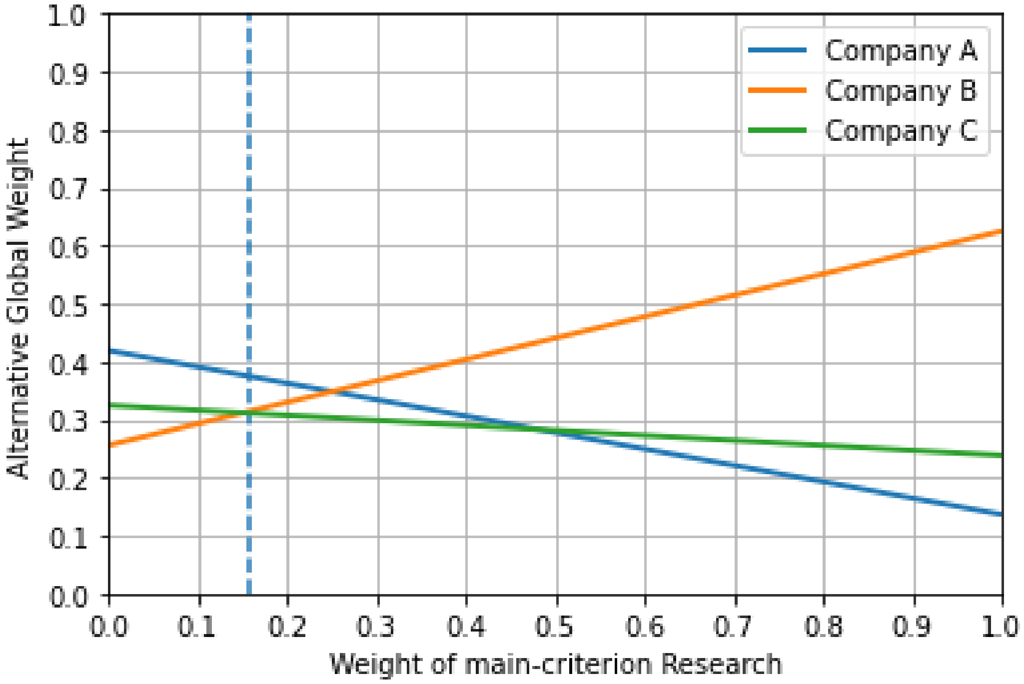
```
Inside None
[[ 1    3   1/3 ]
 [1/3   1   1/7 ]
 [ 3    7    1  ]]
Lambda = 3.007022, CI= 0.003511, CR= 0.006053
Alternative weights= [0.242637 0.087946 0.669417]
Inside None
[[ 1   1/3   5  ]
 [ 3    1    7  ]
 [1/5  1/7   1  ]]
Lambda = 3.064888, CI= 0.032444, CR= 0.055938
Alternative weights= [0.278955 0.649118 0.071927]
Inside None
[[ 1    1    7  ]
 [ 1    1    7  ]
 [1/7  1/7   1  ]]
Lambda = 3.000000, CI= 0.000000, CR= 0.000000
Alternative weights= [0.466667 0.466667 0.066667]
Inside None
[[ 1    7    9  ]
 [1/7   1    2  ]
 [1/9  1/2   1  ]]
Lambda = 3.021730, CI= 0.010865, CR= 0.018733
Alternative weights= [0.792757 0.131221 0.076021]

Results:
  Company A : 0.374467
  Company B : 0.314491
  Company C : 0.311042

Sorted Results:
  Company A : 0.374467
  Company B : 0.314491
  Company C : 0.311042
```

[12]:
```python
# print alternative global weights
print(result)
```

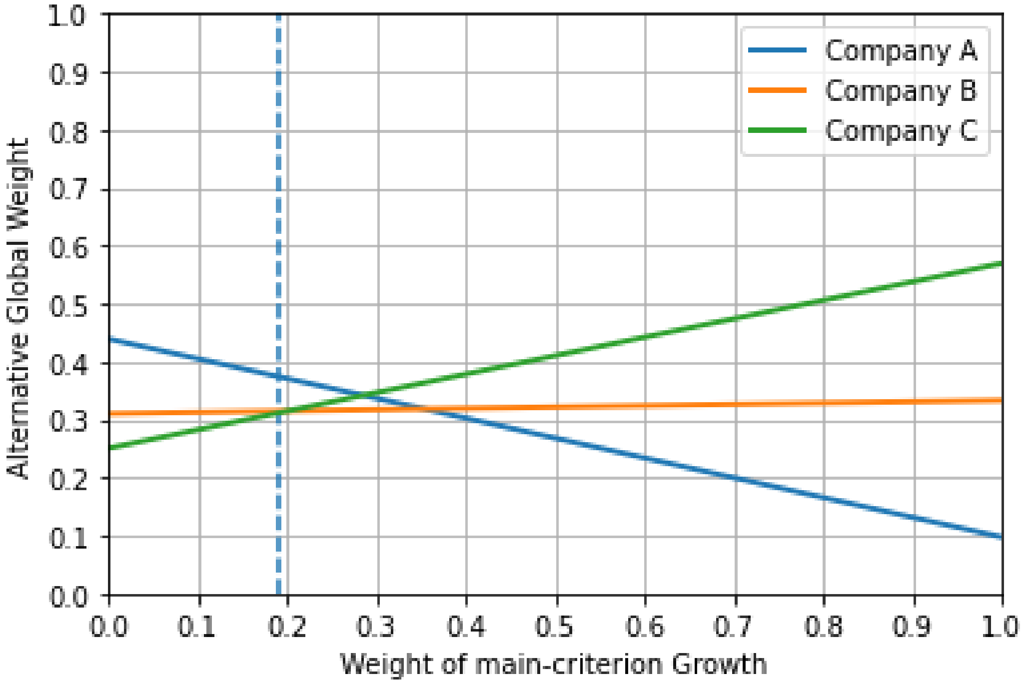
```
{'Company A': 0.3744666462485834, 'Company B': 0.3144914469034947,
 ↪'Company C':
0.31104190684792177}
```

[13]:
```python
# Perform sensitivity analysis
JobSelect.sensit(ymax=1, ystep=0.1)
```

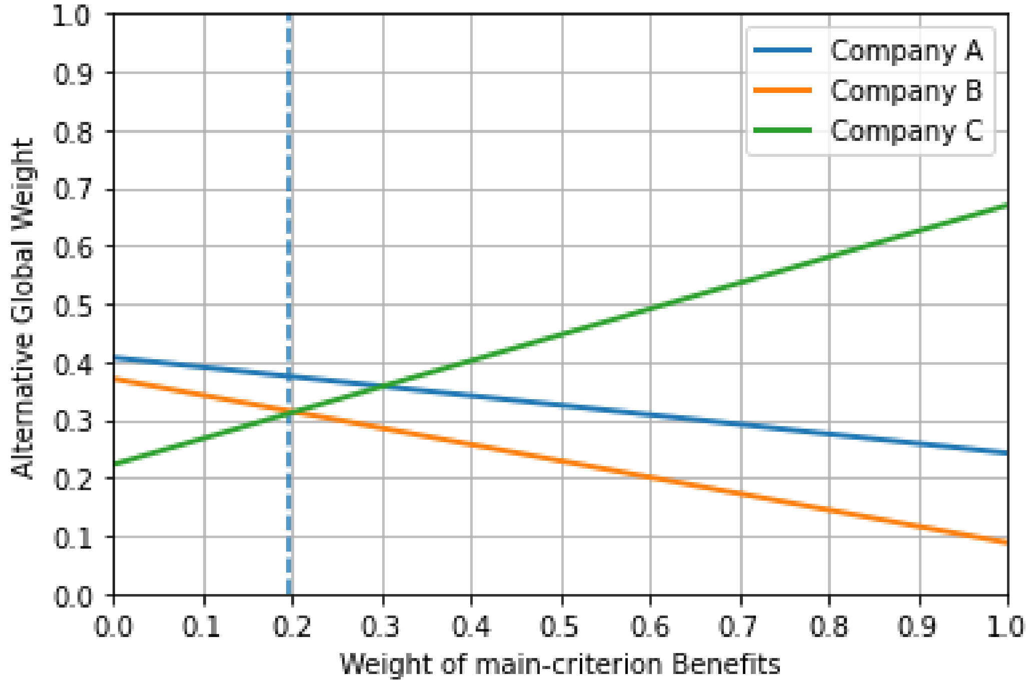
```
Sensitivity Analysis:

Rainbow Diagram for changing weight of main criterion Research
```



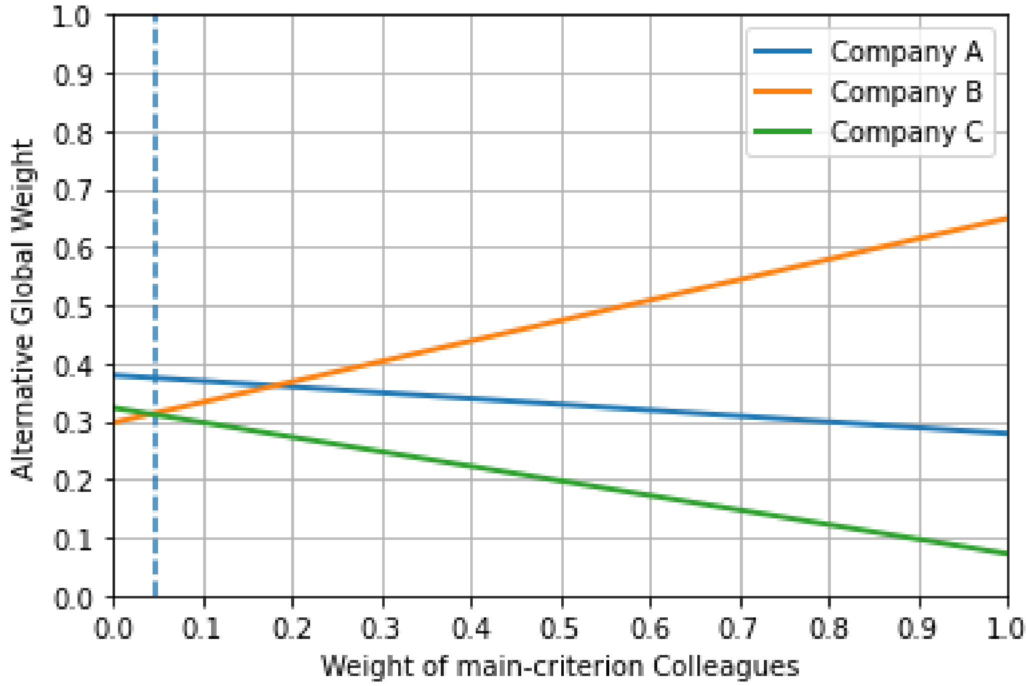
Rainbow Diagram for changing weight of main criterion Growth



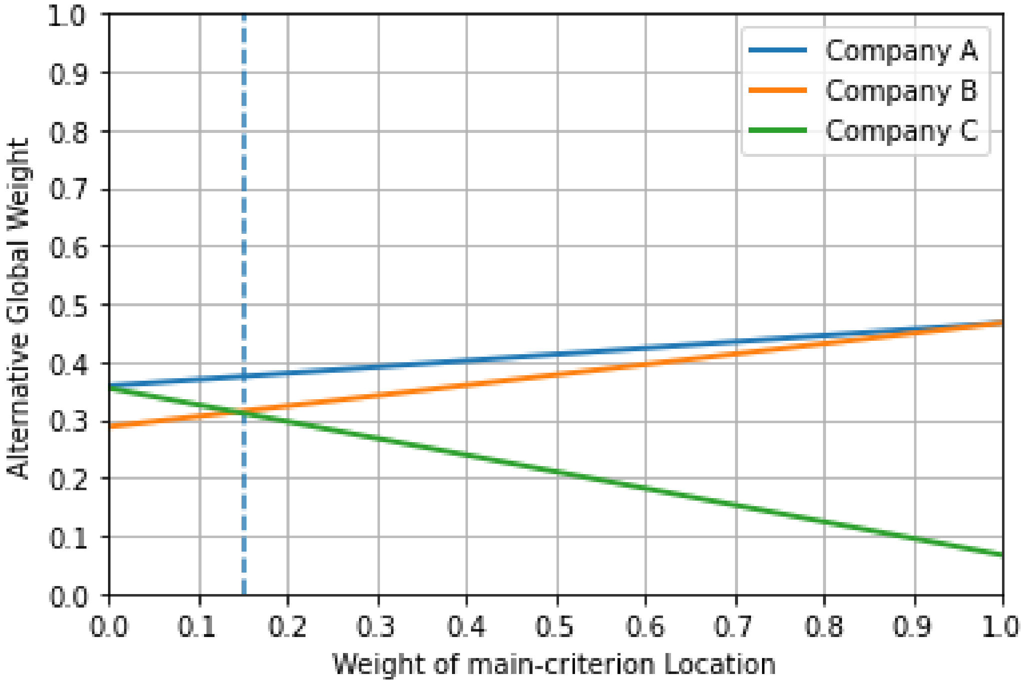
Rainbow Diagram for changing weight of main criterion Benefits



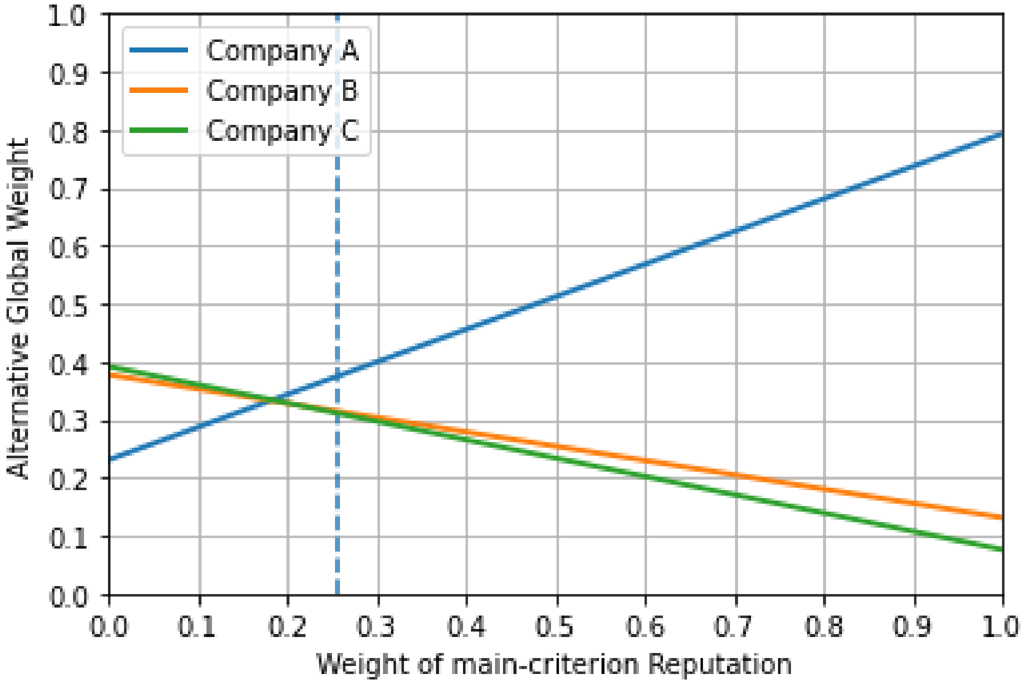
Rainbow Diagram for changing weight of main criterion Colleagues



Rainbow Diagram for changing weight of main criterion Location



Rainbow Diagram for changing weight of main criterion Reputation



[ ]:

11 Class AHPratings_model

11.1 Documentation

[1]:
```
from DecisionAnalysisPy import AHPratings_model
print(AHPratings_model.__doc__)
```

```
Class for AHP Ratings Method model
   Parameters:
     Criteria = dict of main criteria names and matrices
     Ratings = dict of criteria ratings and their matrices
     echo (False) = Bool to show progress of model creation or not
     method ('Algebra') = a valid method for solving AHP matrix

   Methods:
     show_model: Pretty print the model
       Parameter:
         Nil

     evaluate: Evaluate the rated candidates
       Parameter:
         Dict of candidates and their ratings under each criterion
       Return:
         Dict of candidate names and their total scores

     sensit: Perform Sensitivity Analysis with Rainbow Diagrams
       Parameter:
         Dict of candidates and their ratings under each criterion
       Outputs:
         A rainbow diagram for each criterion
```

[ ]:
```

```

11.2 AHP Ratings Model: Employees Evaluation

Source: 9.6.2_AHP_ratings_method_with_AHPratings_model_Class.ipynb

[1]:
```
# 9.6.2_AHP_ratings_method_with_AHPratings_model_Class.ipynb
""" AHP Ratings Method using Class AHPratings_model """
from DecisionAnalysisPy import AHPmatrix
from DecisionAnalysisPy import AHPratings_model
```

[2]:
```
# Define the Main Criteria:
Criteria = {'Names': ['Quality', 'Education', 'Experience',␣
↪'Dependability'],
            'A'    : AHPmatrix([5, 7,  3,
                                3, 1/3,
```

```
                                            1/5 ], upper_triangle=True)
            }
```

[3]:
```
# Define the Criteria ratings and scores
Ratings = { 'Quality' :
                {'Ratings': ['Excellent', 'Good', 'Meet Requirements',
                             'Need Improvements', 'Unsatisfactory'],
                 'A' :      AHPmatrix([1, 5, 7, 9,
                                          3, 5, 7,
                                             3, 5,
                                                2 ],␣
 ↪upper_triangle=True)
                },

            'Education':
                {'Ratings': ['Postgraduate', 'Bachelor', 'Diploma',
                             'Below Dip'],
                 'A':       AHPmatrix([3, 5, 9,
                                          3, 7,
                                             3 ], upper_triangle=True)
                 },

            'Experience':
                {'Ratings': ['> 10 years', '5-10 years', '3-5 years',
                             '1-3 years', '< 1 year'],
                 'A' :      AHPmatrix([1, 3, 5, 7,
                                          2, 3, 5,
                                             2, 3,
                                                3 ],␣
 ↪upper_triangle=True),
                 },

            'Dependability':
                {'Ratings': ['Exceptional', 'Good', 'Satisfactory',␣
 ↪'Poor'],
                 'A' :      AHPmatrix([3, 5, 9,
                                          3, 5,
                                             3 ], upper_triangle=True)
                 }
            }
```

[4]:
```
# Create the AHP Rating Method model
model = AHPratings_model(Criteria, Ratings, echo=True, method='Algebra')
# Show the model created
model.show_model()
```

```
Main Criteria:  ['Quality', 'Education', 'Experience', 'Dependability']
```

```
[[ 1    5    7    3  ]
 [1/5   1    3   1/3 ]
 [1/7  1/3   1   1/5 ]
 [1/3   3    5    1  ]]
  lambda_max = 4.1170,  CI = 0.038994,  CR = 0.043327
  Criteria weights =   0.565009,  0.117504,  0.055285,  0.262201

Criterion 1: Quality
Ratings = ['Excellent', 'Good', 'Meet Requirements', 'Need Improvements',
'Unsatisfactory']
[[ 1    1    5    7    9  ]
 [ 1    1    3    5    7  ]
 [1/5  1/3   1    3    5  ]
 [1/7  1/5  1/3   1    2  ]
 [1/9  1/7  1/5  1/2   1  ]]
 lambda_max = 5.1356,  CI = 0.033899,  CR = 0.030267
 normalized w =   0.428747,  0.337852,  0.136310,  0.060049,  0.037042
 idealized  w =   1.000000,  0.787997,  0.317927,  0.140058,  0.086395

Criterion 2: Education
Ratings = ['Postgraduate', 'Bachelor', 'Diploma', 'Below Dip']
[[ 1    3    5    9  ]
 [1/3   1    3    7  ]
 [1/5  1/3   1    3  ]
 [1/9  1/7  1/3   1  ]]
 lambda_max = 4.0876,  CI = 0.029210,  CR = 0.032456
 normalized w =   0.573455,  0.271227,  0.110233,  0.045086
 idealized  w =   1.000000,  0.472971,  0.192225,  0.078621

Criterion 3: Experience
Ratings = ['> 10 years', '5-10 years', '3-5 years', '1-3 years', '< 1␣
 ↪year']
[[ 1    1    3    5    7  ]
 [ 1    1    2    3    5  ]
 [1/3  1/2   1    2    3  ]
 [1/5  1/3  1/2   1    3  ]
 [1/7  1/5  1/3  1/3   1  ]]
 lambda_max = 5.0872,  CI = 0.021807,  CR = 0.019471
 normalized w =   0.393131,  0.304538,  0.153971,  0.099081,  0.049280
 idealized  w =   1.000000,  0.774648,  0.391653,  0.252030,  0.125353

Criterion 4: Dependability
Ratings = ['Exceptional', 'Good', 'Satisfactory', 'Poor']
[[ 1    3    5    9  ]
 [1/3   1    3    5  ]
 [1/5  1/3   1    3  ]
 [1/9  1/5  1/3   1  ]]
 lambda_max = 4.0763,  CI = 0.025431,  CR = 0.028257
```

```
 normalized w =   0.580592,  0.255358,  0.114114,  0.049937
 idealized  w =   1.000000,  0.439823,  0.196547,  0.086010

AHP Ratings Method Model created.

AHP Ratings Method Model:
  Main Criteria = ['Quality', 'Education', 'Experience', 'Dependability']
[[ 1    5    7    3  ]
 [1/5   1    3   1/3 ]
 [1/7  1/3   1   1/5 ]
 [1/3   3    5    1  ]]
  Lambda_max = 4.1170,   CI = 0.0390   CR = 0.0433
  w =   0.565009,  0.117504,  0.055285,  0.262201

Criterion 1: Quality
Ratings = ['Excellent', 'Good', 'Meet Requirements', 'Need Improvements',
'Unsatisfactory']
[[ 1    1    5    7    9  ]
 [ 1    1    3    5    7  ]
 [1/5  1/3   1    3    5  ]
 [1/7  1/5  1/3   1    2  ]
 [1/9  1/7  1/5  1/2   1  ]]
 Lambda_max = 5.1356   CI = 0.0339   CR = 0.0303
 normalized w =   0.428747,  0.337852,  0.136310,  0.060049,  0.037042
 idealized  w =   1.000000,  0.787997,  0.317927,  0.140058,  0.086395

Criterion 2: Education
Ratings = ['Postgraduate', 'Bachelor', 'Diploma', 'Below Dip']
[[ 1    3    5    9  ]
 [1/3   1    3    7  ]
 [1/5  1/3   1    3  ]
 [1/9  1/7  1/3   1  ]]
 Lambda_max = 4.0876   CI = 0.0292   CR = 0.0325
 normalized w =   0.573455,  0.271227,  0.110233,  0.045086
 idealized  w =   1.000000,  0.472971,  0.192225,  0.078621

Criterion 3: Experience
Ratings = ['> 10 years', '5-10 years', '3-5 years', '1-3 years', '< 1
 year']
[[ 1    1    3    5    7  ]
 [ 1    1    2    3    5  ]
 [1/3  1/2   1    2    3  ]
 [1/5  1/3  1/2   1    3  ]
 [1/7  1/5  1/3  1/3   1  ]]
 Lambda_max = 5.0872   CI = 0.0218   CR = 0.0195
 normalized w =   0.393131,  0.304538,  0.153971,  0.099081,  0.049280
 idealized  w =   1.000000,  0.774648,  0.391653,  0.252030,  0.125353
```

```
Criterion 4: Dependability
Ratings = ['Exceptional', 'Good', 'Satisfactory', 'Poor']
[[ 1    3    5    9  ]
 [1/3   1    3    5  ]
 [1/5  1/3   1    3  ]
 [1/9  1/5  1/3   1  ]]
 Lambda_max = 4.0763   CI = 0.0254   CR = 0.0283
 normalized w =   0.580592,  0.255358,  0.114114,  0.049937
 idealized  w =   1.000000,  0.439823,  0.196547,  0.086010
```

[5]:
```
# Employees and their ratings for each criterion
Employees = {'John Lim':
                 {'Quality'      : 'Good',
                  'Education'    : 'Postgraduate',
                  'Experience'   : '3-5 years',
                  'Dependability': 'Satisfactory'
                  },
             'Tan Ah Huay':
                 {'Quality'      : 'Meet Requirements',
                  'Education'    : 'Diploma',
                  'Experience'   : '> 10 years',
                  'Dependability': 'Good'
                  },
             'Chow Ah Beng':
                 {'Quality'      : 'Need Improvements',
                  'Education'    : 'Below Dip',
                  'Experience'   : '1-3 years',
                  'Dependability': 'Poor'
                  },
             'Mary Lau':
                 {'Quality'      : 'Good',
                  'Education'    : 'Bachelor',
                  'Experience'   : '< 1 year',
                  'Dependability': 'Exceptional'
                 },
             'Harry Lee':
                 {'Quality'      : 'Excellent',
                  'Education'    : 'Diploma',
                  'Experience'   : '5-10 years',
                  'Dependability': 'Good'
                  }
            }
```

[6]:
```
# Evaluate employees
model.evaluate(Employees)
```

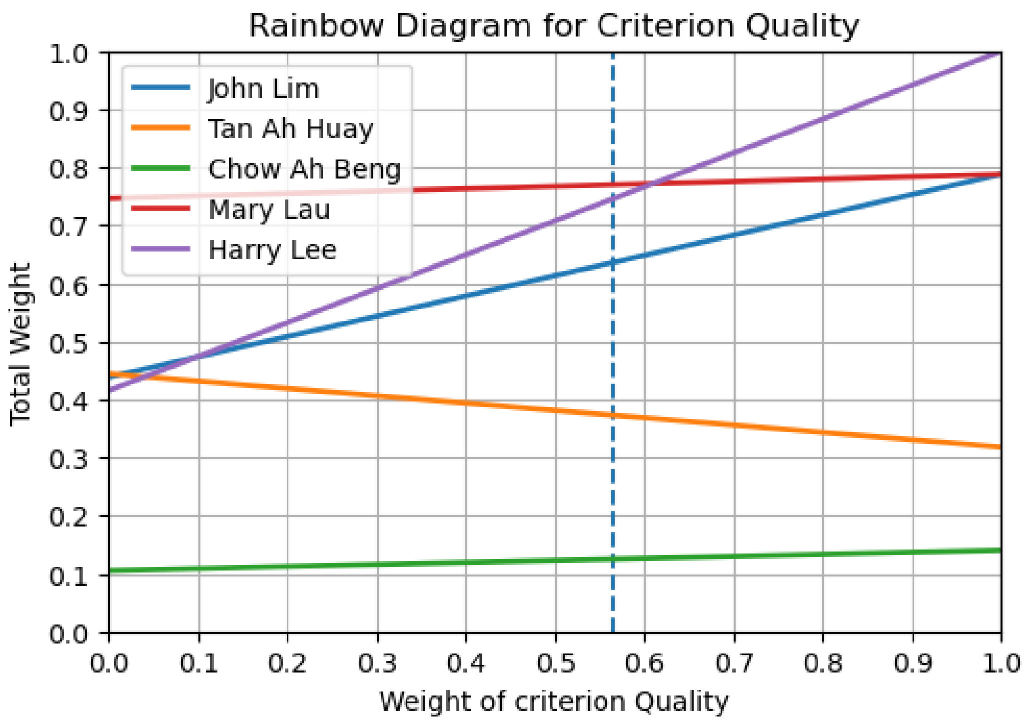
```
Overall Scores (sorted):
  Mary Lau     : 0.769933
```

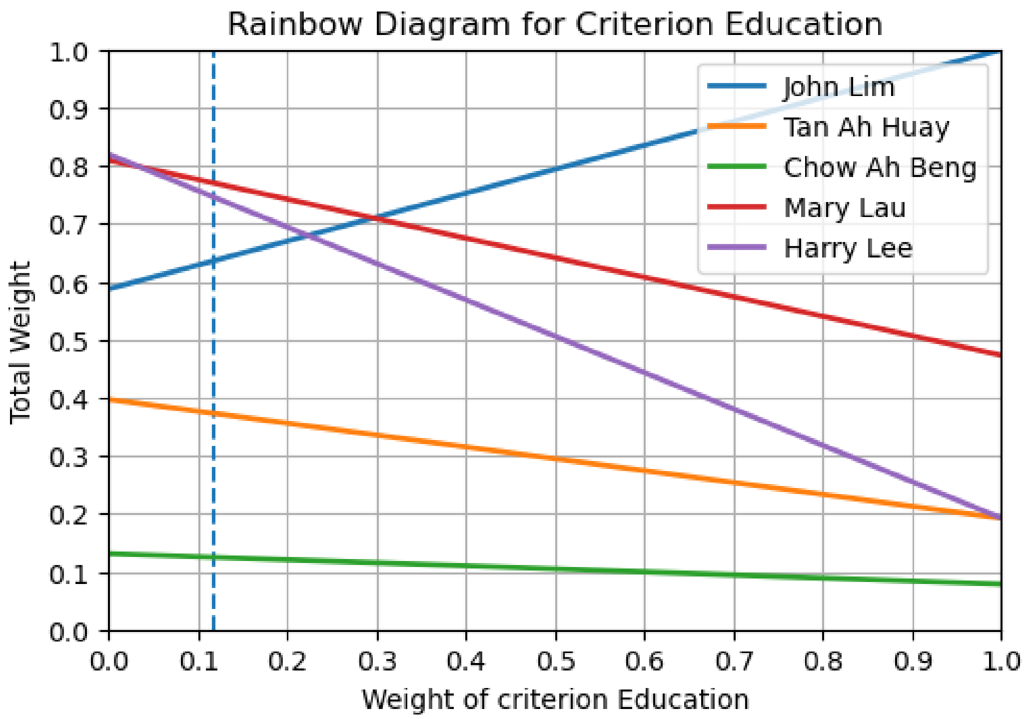
```
Harry Lee     : 0.745745
John Lim      : 0.635918
Tan Ah Huay   : 0.372826
Chow Ah Beng  : 0.124858
```

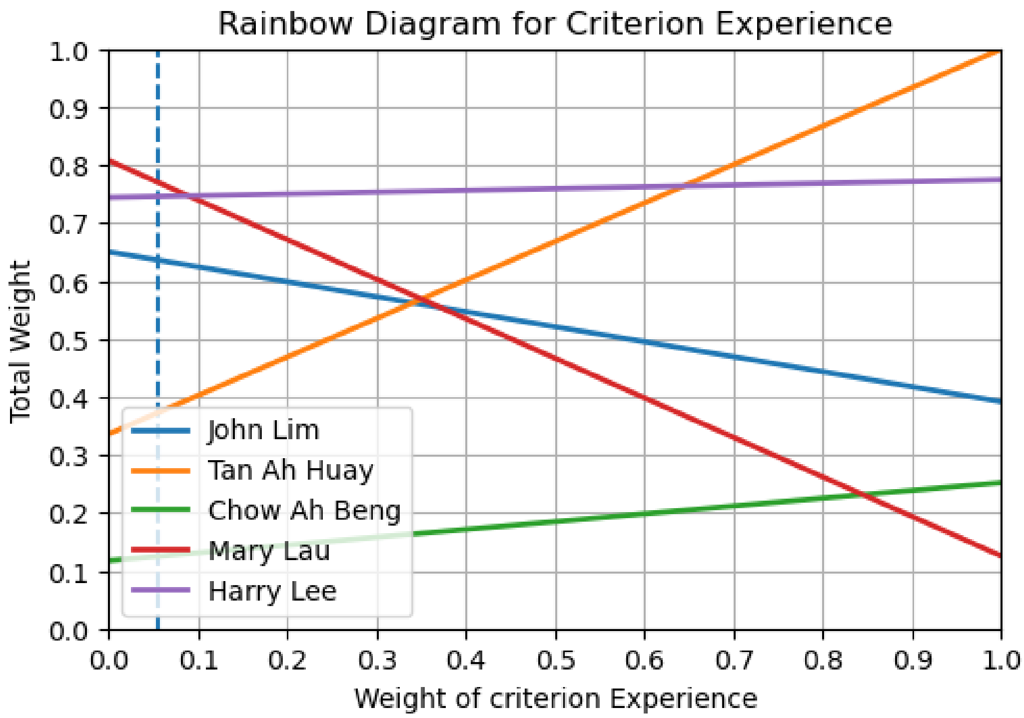
[6]:
```
{'John Lim': 0.6359175130542506,
 'Tan Ah Huay': 0.3728263943688019,
 'Chow Ah Beng': 0.12485779987990897,
 'Mary Lau': 0.7699330723103244,
 'Harry Lee': 0.7457452132146091}
```

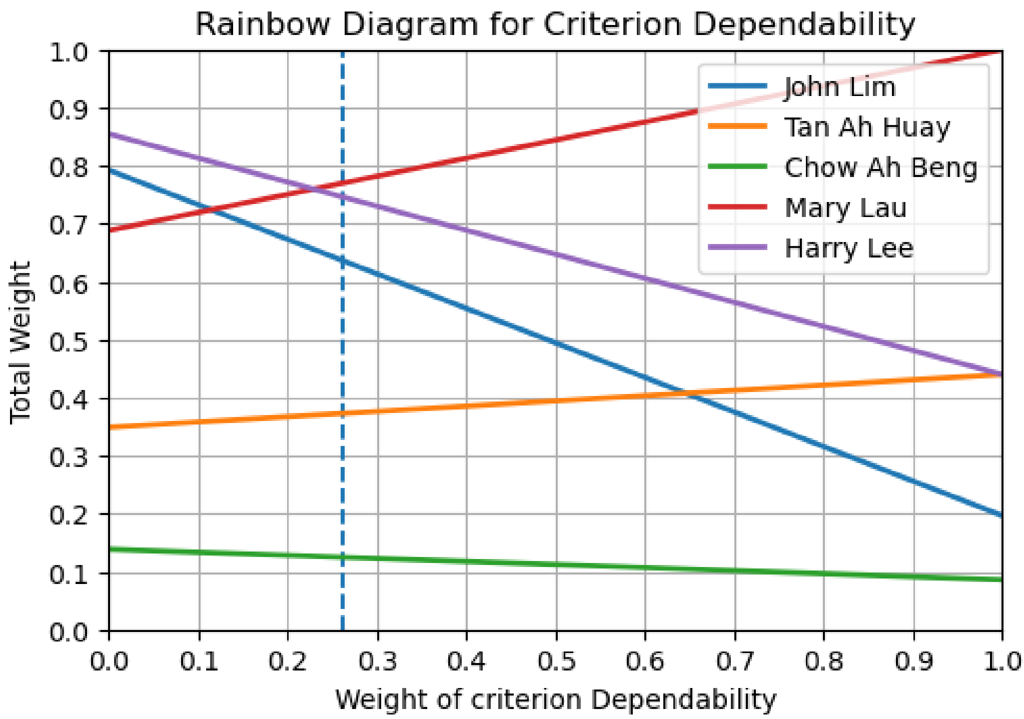
[7]:
```
model.sensit(Employees)
```

```
Sensivity Analysis:
```




Rainbow Diagram for Criterion Education
Total Weight
Weight of criterion Education
John Lim
Tan Ah Huay
Chow Ah Beng
Mary Lau
Harry Lee
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


Rainbow Diagram for Criterion Experience
Total Weight
Weight of criterion Experience
John Lim
Tan Ah Huay
Chow Ah Beng
Mary Lau
Harry Lee
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0




```
[ ]:
```

11.3 Problem 9.2: John and Bill Problem

Source: 9_Problem_9.2_AHPratings_model_Class.ipynb

```
[1]: """ Solutions to Problem 9.2 using AHPratings_model Class """
     from DecisionAnalysisPy import AHPmatrix
     from DecisionAnalysisPy import AHPratings_model
```

```
[2]: # Define the Main Criteria:
     Criteria = {'Names':
       ↪['Dependability','Qualification','Experience','Quality'],
                 'A'     : AHPmatrix([2, 3, 4,
                                         2, 3,
                                            2 ], upper_triangle=True)
                }
```

```
[3]: # Define the Criteria ratings and scores
     Ratings = { 'Dependability':
                   {'Ratings': ['Outstanding','Average','Unsatisfactory'],
                    'A' :      AHPmatrix([3, 7,
                                             3 ], upper_triangle=True)
                   },
```

```
            'Qualification':
               {'Ratings': ['Postgraduate','Graduate','Non-graduate'],
                'A' :      AHPmatrix([3, 5,
                                          3 ], upper_triangle=True)
               },

            'Experience':
               {'Ratings': ['Exceptional','Average','Little'],
                'A' :      AHPmatrix([5, 9,
                                          3 ], upper_triangle=True)
               },

            'Quality':
               {'Ratings': ['Outstanding','Average','Below average'],
                'A' :      AHPmatrix([5, 9,
                                          3 ], upper_triangle=True)
               }
          }
```

[4]:
```
# Create the AHP Ratings Method model
model = AHPratings_model(Criteria, Ratings, echo=False,␣
 ↪method='GenEigen')
# Show the model created
model.show_model()
```

```
AHP Ratings Method Model:
  Main Criteria = ['Dependability', 'Qualification', 'Experience',␣
 ↪'Quality']
[[ 1    2    3    4  ]
 [1/2   1    2    3  ]
 [1/3  1/2   1    2  ]
 [1/4  1/3  1/2   1  ]]
  Lambda_max = 4.0310,   CI = 0.0103   CR = 0.0115
  w =   0.467296,  0.277181,  0.160088,  0.095435

Criterion 1: Dependability
Ratings = ['Outstanding', 'Average', 'Unsatisfactory']
[[ 1    3    7  ]
 [1/3   1    3  ]
 [1/7  1/3   1  ]]
 Lambda_max = 3.0070   CI = 0.0035   CR = 0.0061
 normalized w =   0.669417,  0.242637,  0.087946
 idealized  w =   1.000000,  0.362460,  0.131377

Criterion 2: Qualification
Ratings = ['Postgraduate', 'Graduate', 'Non-graduate']
[[ 1    3    5  ]
```

```
 [1/3   1    3  ]
 [1/5  1/3   1  ]]
 Lambda_max = 3.0385   CI = 0.0193   CR = 0.0332
 normalized w =   0.636986,  0.258285,  0.104729
 idealized  w =   1.000000,  0.405480,  0.164414

Criterion 3: Experience
Ratings = ['Exceptional', 'Average', 'Little']
[[ 1    5    9  ]
 [1/5   1    3  ]
 [1/9  1/3   1  ]]
 Lambda_max = 3.0291   CI = 0.0145   CR = 0.0251
 normalized w =   0.751405,  0.178178,  0.070418
 idealized  w =   1.000000,  0.237126,  0.093715

Criterion 4: Quality
Ratings = ['Outstanding', 'Average', 'Below average']
[[ 1    5    9  ]
 [1/5   1    3  ]
 [1/9  1/3   1  ]]
 Lambda_max = 3.0291   CI = 0.0145   CR = 0.0251
 normalized w =   0.751405,  0.178178,  0.070418
 idealized  w =   1.000000,  0.237126,  0.093715
```

[5]:

```
Candidates = {'John':
                 {'Dependability': 'Average',
                  'Qualification': 'Graduate',
                  'Experience'   : 'Average',
                  'Quality'      : 'Outstanding'
                 },

              'Bill':
                 {'Dependability': 'Outstanding',
                  'Qualification': 'Non-graduate',
                  'Experience'   : 'Exceptional',
                  'Quality'      : 'Average'
                 }
              }
```

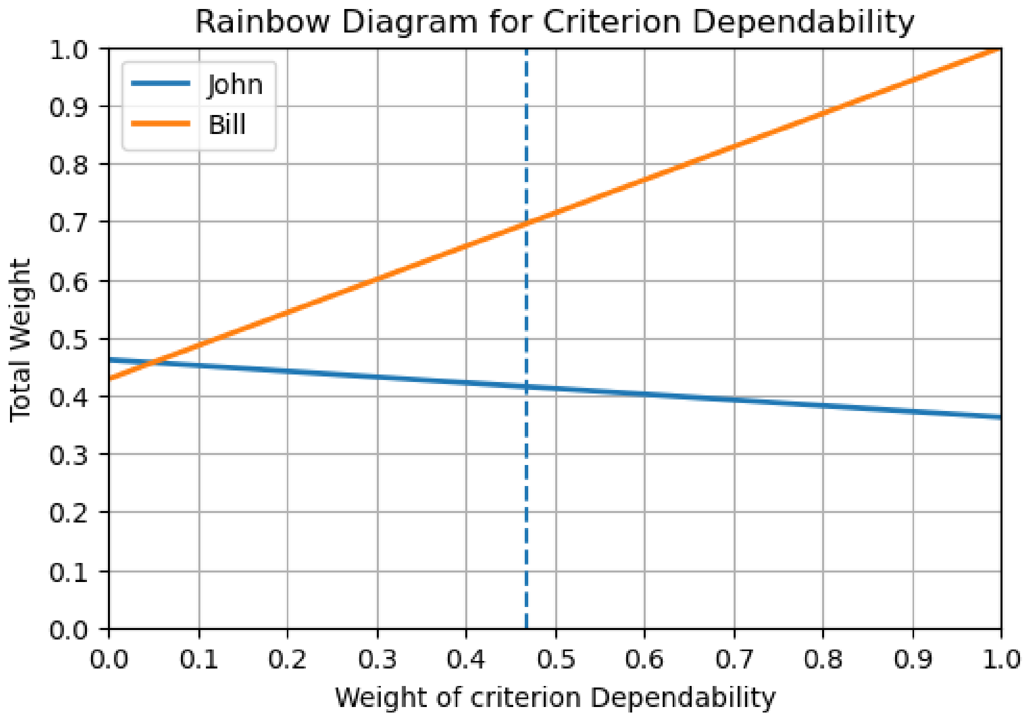
[6]:

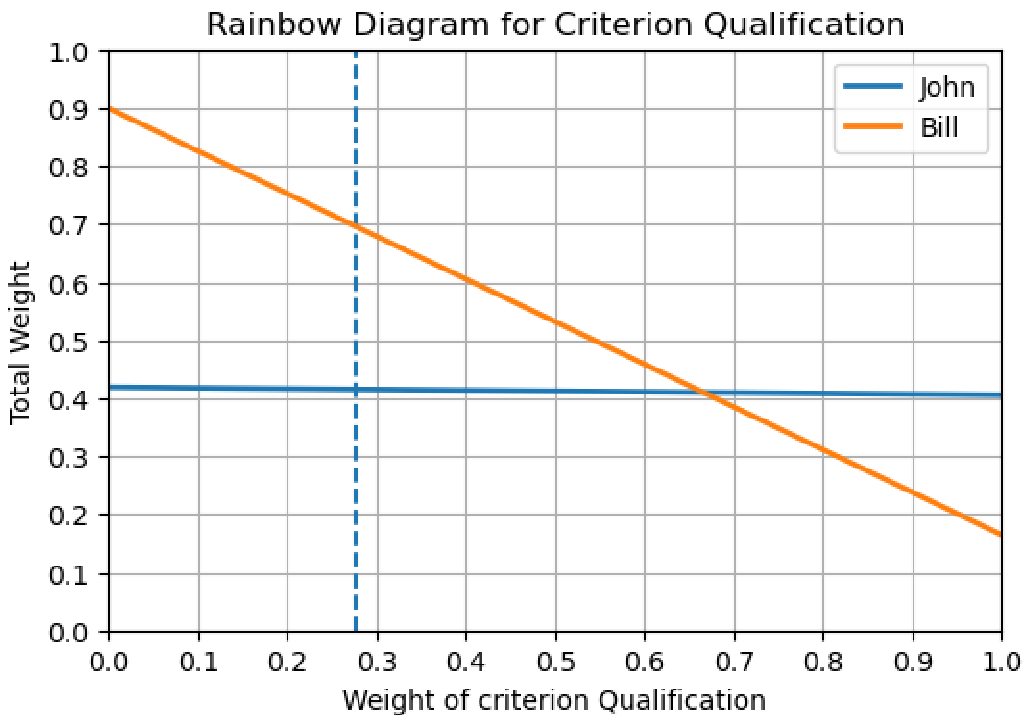
```
# Evaluate employees
scores = model.evaluate(Candidates)
print(f"{max(scores, key=scores.get)} should get higher pay increase")
```

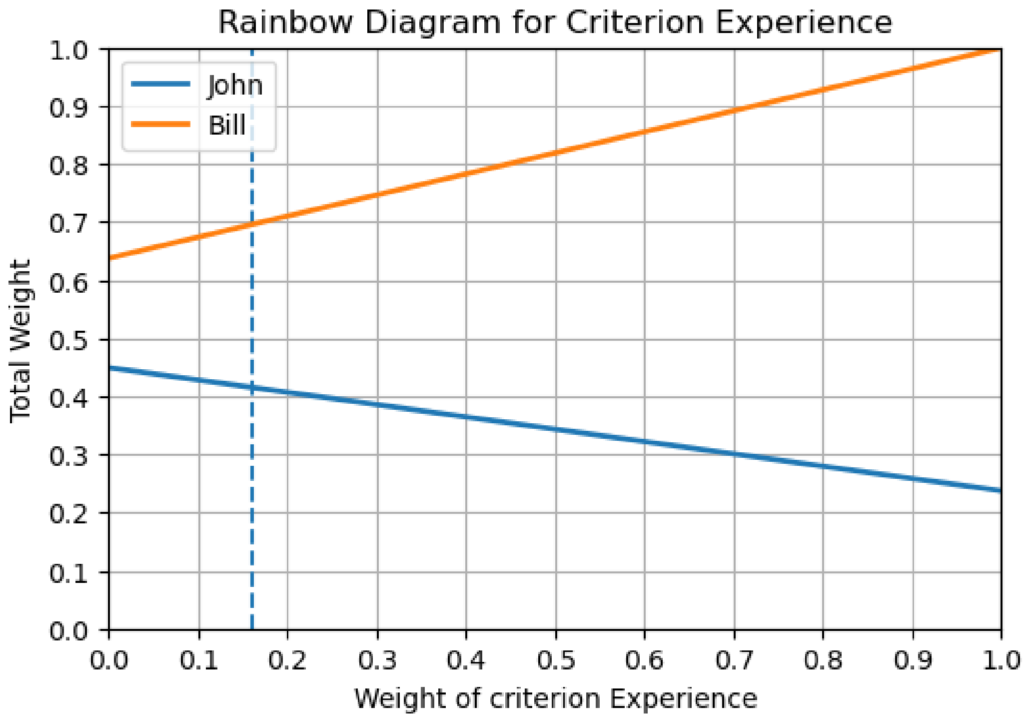
```
Overall Scores (sorted):
  Bill          : 0.695587
  John          : 0.415164
Bill should get higher pay increase
```

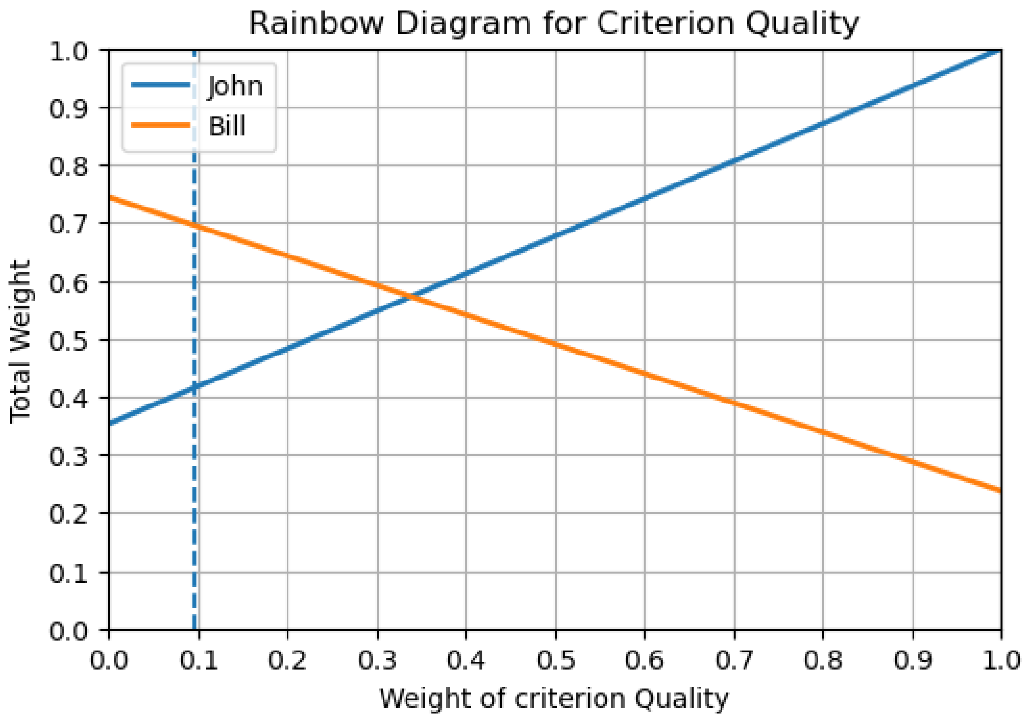
[7]:
```
model.sensit(Candidates)
```

```
Sensivity Analysis:
```






Rainbow Diagram for Criterion Experience
John
Bill
Total Weight
Weight of criterion Experience
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


Rainbow Diagram for Criterion Quality
John
Bill
Total Weight
Weight of criterion Quality
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

12 Class Cost_Effective_Analysis

12.1 Documentation

[1]:
```
from DecisionAnalysisPy import Cost_Effective_Analysis
```

[2]:
```
print(Cost_Effective_Analysis.__doc__)
```

```
 Performance cost-effective analysis and plot efficient frontier
    Parameters:
      Attributes: ['Cost attribute name', 'Effectivess attribute name']
      Alternatives:
          dict as { alternative_name : (cost, effectiveness)}

     Methods:
       get_efficient_set():
           Returns a sub-dictionary of alternatives that are efficient

       plot_efficient_frontier(xlim, ylim, figsize=None, dpi=100):
          Plot the efficient points and frontier
            xlim = (xmin, xmax) values to plot
            ylim = (ymin, ymax) values to plot
            figsize = figsize to plot
            dpi = dpi to plot
```

[ ]:
```

```

12.2 Drug Counselling Center Relocation Problem: Complete AHP and Cost Effectiveness Analysis

Source: 10.3_Center_Relocation_Problem_complete_cost_effectiveness_analysis.ipynb

[1]:
```
""" Drug Counseling Center Relocation Problem:
    (a) Effectivess evaluation with AHP model using AHP4Lmodel Class
    (b) Cost-effectiveness and efficient frontier analysis using
            Cost_Effective_Analysis Class"""
from DecisionAnalysisPy import AHP4Lmodel, Cost_Effective_Analysis
import numpy as np
```

[2]:
```
# Site Effectiveness Evaluation with AHP
goal = "Best Site for Relocation"
alternatives = ["Site 1", "Site 2", "Site 3", "Site 4", "Site 5", "Site␣
 ↪6"]
main_cr = ["Good conditions for staff",
            "Easy access for clients",
            "Suitability of space for center's functions",
            "Administrative convenience" ]
```

```
main_cr_mat = np.array([2, 2, 3,
                           1, 2,
                              1 ])
```

[3]:
```
# Containers for data
sub_cr = []
sub_cr_mats = []
alt_cr_mats = []

# Main Criterion 1: "Good Conditions for staff"
# List of subcriteria for Criterion 1
sub_cr.append(["Office size",
               "Convenience of staff commuting",
                "Office attractiveness",
                "Office privacy",
                "Availability of parking" ])

# Pairwise comparison of subcriteria for Criterion 1
sub_cr_mats.append(np.array([2, 3, 3, 3,
                                2, 2, 2,
                                   1, 1,
                                      1 ]))

# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_cr_mats.append([np.array([2, 9, 2,  9,  2,
                                 5, 1,  5,  1,
                                   1/5, 1, 1/4,
                                        5,  1,
                                           1/4 ]),
                    np.array([2, 1/2, 5, 9,  2,
                                 1/3, 3, 6,          1,
                                      9, 9,  3,
                                         2, 1/3,
                                            1/6 ]),
                    np.array([1/3, 1/2, 3, 1/3, 1/3,
                                   1,  8,  1,   1,
                                       7,  1,   1,
                                          1/9, 1/8,
                                                1 ]),
                    np.array([3, 2, 9,  3,   2,
                                 1, 3,  1,  1/2,
                                    4,  1,   1,
                                       1/3, 1/5,
                                             1 ]),

                    np.array([1/6, 1/3, 1, 1/9, 1/5,
```

```
                                2,  6, 1/2,  1,
                                    3,  1/3, 1/2,
                                        1/9, 1/5,
                                              2  ])])
```

[4]:
```
# Main Criterion 2: "Easy access for clients"
# List of subcriteria for Criterion 2
sub_cr.append(["Closeness to client's homes",
               "Access to public transportation" ])

# Pairwise comparison of subcriteria for Criterion 2
sub_cr_mats.append(np.array([ 1 ]))

# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_cr_mats.append([ np.array([1, 3, 1/2, 1/3,  1,
                              3, 1/2, 1/3,  1,
                                  1/5, 1/9, 1/3,
                                      1/2,  2,
                                            3 ]),
                     np.array([1, 1, 1, 7,  1,
                              1, 1, 7,  1,
                                 2, 9,  1,
                                    5,  1,
                                       1/7 ]) ])
```

[5]:
```
# Main Criterion 3: "Suitability of space for for center's functions"
# List of subcriteria for Criterion 3
sub_cr.append(["Number and suitability of counseling rooms",
               "Number and suitability of conference rooms",
               "Suitability of reception and waiting area" ])

# Pairwise comparison of subcriteria for Criterion 3
sub_cr_mats.append(np.array([ 2, 2, 2 ]))

# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_cr_mats.append([np.array([1/8, 2, 1/5, 1/9, 1/5,
                                   9,  2,   1,   2,
                                       1/9, 1/9, 1/9,
                                            1/2,  1,
                                                  2 ]),
                    np.array([1, 6, 6, 1/2,  2,
                                 5,  5, 1/2, 2,
                                    1, 1/9, 1/3,
                                        1/9, 1/3,
                                              3 ]),
                        np.array([1, 1, 5,  2,   2,
                                     1, 4, 1/2,  1,
                                        5, 1/2,  2,
```

```
                                                1/9, 1/3,
                                                      3  ])] )
```

```
[6]: # Main Criterion 4: "Administrative Convenience"
# List of subcriteria for Criterion 4
sub_cr.append(["Adequacy of space for admin work",
               "Flexibility of the space layout" ])

# Pairwise comparison of subcriteria for Criterion 4
sub_cr_mats.append(np.array([ 2 ]))

# Pairwise comparison of alternatives w.r.t. each subcriterion
alt_cr_mats.append([np.array([1/7, 1/5, 1/9, 1/5, 1/6,
                                    1,   1,   1,   1,
                                        1/2,  1,   1,
                                              2,   2,
                                                   1 ]),
                    np.array([1/4, 1/5,      1/9, 1,       1/4,
                                    1,   2,  4,  1,
                                        1/2, 5,  1,
                                             9,  1,
                                                1/4 ]) ])

# End of 4-Level AHP Model Definition and Data #
```

```
[7]: # Create an instance of a 4-Level AHP model
DrugCenter = AHP4Lmodel(goal, main_cr, main_cr_mat, sub_cr, sub_cr_mats,
                            alternatives, alt_cr_mats)
```

```
[8]: # Show the AHP model
# DrugCenter.model()
```

```
[9]: # Compute global weights
site_global_wt = DrugCenter.solve(method="Algebra")
```

```
Goal: Best Site for Relocation
Alternatives: ['Site 1', 'Site 2', 'Site 3', 'Site 4', 'Site 5', 'Site␣
 ↪6']
Main Criteria: ['Good conditions for staff', 'Easy access for clients',
"Suitability of space for center's functions", 'Administrative␣
 ↪convenience']

Pairwise comparison of Main Criteria w.r.t. Goal Best Site for␣
 ↪Relocation:
[[ 1    2    2    3  ]
 [1/2   1    1    2  ]
 [1/2   1    1    1  ]
```

```
 [1/3  1/2   1    1  ]]
Lambda = 4.045819, CI= 0.015273, CR= 0.016970
Main criteria weights= [0.425784 0.231236 0.194548 0.148431]

Main Criteria 1: Good conditions for staff
Sub Criteria: ['Office size', 'Convenience of staff commuting', 'Office
attractiveness', 'Office privacy', 'Availability of parking']
Pairwise comparison of Sub-Criteria for Good conditions for staff:
[[ 1    2    3    3    3  ]
 [1/2   1    2    2    2  ]
 [1/3  1/2   1    1    1  ]
 [1/3  1/2   1    1    1  ]
 [1/3  1/2   1    1    1  ]]
Lambda = 5.009960, CI= 0.002490, CR= 0.002223
Sub-citeria weights= [0.394537 0.23431  0.123718 0.123718 0.123718]

Pairwise comparison of Alternatives wrt Office size
[[ 1    2    9    2    9    2  ]
 [1/2   1    5    1    5    1  ]
 [1/9  1/5   1   1/5   1   1/4 ]
 [1/2   1    5    1    5    1  ]
 [1/9  1/5   1   1/5   1   1/4 ]
 [1/2   1    4    1    4    1  ]]
Lambda = 6.007543, CI= 0.001509, CR= 0.001217
Alternative weights= [0.365332 0.189325 0.040006 0.189325 0.040006 0.
 ↪176007]

Pairwise comparison of Alternatives wrt Convenience of staff commuting
[[ 1    2   1/2   5    9    2  ]
 [1/2   1   1/3   3    6    1  ]
 [ 2    3    1    9    9    3  ]
 [1/5  1/3  1/9   1    2   1/3 ]
 [1/9  1/6  1/9  1/2   1   1/6 ]
 [1/2   1   1/3   3    6    1  ]]
Lambda = 6.066245, CI= 0.013249, CR= 0.010685
Alternative weights= [0.246725 0.139716 0.397855 0.047928 0.028059 0.
 ↪139716]

Pairwise comparison of Alternatives wrt Office attractiveness
[[ 1   1/3  1/2   3   1/3  1/3 ]
 [ 3    1    1    8    1    1  ]
 [ 2    1    1    7    1    1  ]
 [1/3  1/8  1/7   1   1/9  1/8 ]
 [ 3    1    1    9    1    1  ]
 [ 3    1    1    8    1    1  ]]
Lambda = 6.018727, CI= 0.003745, CR= 0.003020
Alternative weights= [0.082567 0.225852 0.207524 0.027744 0.230461 0.
 ↪225852]
```

```
Pairwise comparison of Alternatives wrt Office privacy
[[ 1    3    2    9    3    2  ]
 [1/3   1    1    3    1   1/2 ]
 [1/2   1    1    4    1    1  ]
 [1/9  1/3  1/4   1   1/3  1/5 ]
 [1/3   1    1    3    1    1  ]
 [1/2   2    1    5    1    1  ]]
Lambda = 6.057965, CI= 0.011593, CR= 0.009349
Alternative weights= [0.360272 0.123617 0.155182 0.040073 0.138655 0.
 ↪182202]

Pairwise comparison of Alternatives wrt Availability of parking
[[ 1   1/6  1/3   1   1/9  1/5 ]
 [ 6    1    2    6   1/2   1  ]
 [ 3   1/2   1    3   1/3  1/2 ]
 [ 1   1/6  1/3   1   1/9  1/5 ]
 [ 9    2    3    9    1    2  ]
 [ 5    1    2    5   1/2   1  ]]
Lambda = 6.013523, CI= 0.002705, CR= 0.002181
Alternative weights= [0.039471 0.219426 0.114984 0.039471 0.380349 0.
 ↪206299]

Main Criteria 2: Easy access for clients
Sub Criteria: ["Closeness to client's homes", 'Access to public␣
 ↪transportation']
Pairwise comparison of Sub-Criteria for Easy access for clients:
[[ 1    1  ]
 [ 1    1  ]]
Lambda = 2.000000, CI= 0.000000, CR= 0.000000
Sub-citeria weights= [0.5 0.5]

Pairwise comparison of Alternatives wrt Closeness to client's homes
[[ 1    1    3   1/2  1/3   1  ]
 [ 1    1    3   1/2  1/3   1  ]
 [1/3  1/3   1   1/5  1/9  1/3 ]
 [ 2    2    5    1   1/2   2  ]
 [ 3    3    9    2    1    3  ]
 [ 1    1    3   1/2  1/3   1  ]]
Lambda = 6.009998, CI= 0.002000, CR= 0.001613
Alternative weights= [0.119713 0.119713 0.041136 0.222109 0.377617 0.
 ↪119713]

Pairwise comparison of Alternatives wrt Access to public transportation
[[ 1    1    1    1    7    1  ]
 [ 1    1    1    1    7    1  ]
 [ 1    1    1    2    9    1  ]
 [ 1    1   1/2   1    5    1  ]
```

```
 [1/7  1/7  1/9  1/5   1   1/7 ]
 [ 1    1    1    1    7    1  ]]
Lambda = 6.055665, CI= 0.011133, CR= 0.008978
Alternative weights= [0.192769 0.192769 0.229167 0.164636 0.027891 0.
 ↪192769]

Main Criteria 3: Suitability of space for center's functions
Sub Criteria: ['Number and suitability of counseling rooms', 'Number and
suitability of conference rooms', 'Suitability of reception and waiting␣
 ↪area']
Pairwise comparison of Sub-Criteria for Suitability of space for center's
functions:
[[ 1    2    2  ]
 [1/2   1    2  ]
 [1/2  1/2   1  ]]
Lambda = 3.053622, CI= 0.026811, CR= 0.046225
Sub-citeria weights= [0.493386 0.310814 0.1958   ]

Pairwise comparison of Alternatives wrt Number and suitability of␣
 ↪counseling
rooms
[[ 1   1/8   2   1/5  1/9  1/5 ]
 [ 8    1    9    2    1    2  ]
 [1/2  1/9   1   1/9  1/9  1/9 ]
 [ 5   1/2   9    1   1/2   1  ]
 [ 9    1    9    2    1    2  ]
 [ 5   1/2   9    1   1/2   1  ]]
Lambda = 6.081230, CI= 0.016246, CR= 0.013102
Alternative weights= [0.036847 0.295021 0.024159 0.171447 0.30108  0.
 ↪171447]

Pairwise comparison of Alternatives wrt Number and suitability of␣
 ↪conference
rooms
[[ 1    1    6    6   1/2   2  ]
 [ 1    1    5    5   1/2   2  ]
 [1/6  1/5   1    1   1/9  1/3 ]
 [1/6  1/5   1    1   1/9  1/3 ]
 [ 2    2    9    9    1    3  ]
 [1/2  1/2   3    3   1/3   1  ]]
Lambda = 6.013523, CI= 0.002705, CR= 0.002181
Alternative weights= [0.219426 0.206299 0.039471 0.039471 0.380349 0.
 ↪114984]

Pairwise comparison of Alternatives wrt Suitability of reception and␣
 ↪waiting
area
[[ 1    1    1    5    2    2  ]
```

```
 [ 1    1    1    4   1/2   1  ]
 [ 1    1    1    5   1/2   2  ]
 [1/5  1/4  1/5   1   1/9  1/3 ]
 [1/2   2    2    9    1    3  ]
 [1/2   1   1/2   3   1/3   1  ]]
Lambda = 6.240366, CI= 0.048073, CR= 0.038769
Alternative weights= [0.245092 0.155641 0.178851 0.036349 0.275579 0.
 ↪108488]

Main Criteria 4: Administrative convenience
Sub Criteria: ['Adequacy of space for admin work', 'Flexibility of the␣
 ↪space
layout']
Pairwise comparison of Sub-Criteria for Administrative convenience:
[[ 1    2  ]
 [1/2   1  ]]
Lambda = 2.000000, CI= 0.000000, CR= 0.000000
Sub-citeria weights= [0.666667 0.333333]

Pairwise comparison of Alternatives wrt Adequacy of space for admin work
[[ 1   1/7  1/5  1/9  1/5  1/6 ]
 [ 7    1    1    1    1    1  ]
 [ 5    1    1   1/2   1    1  ]
 [ 9    1    2    1    2    2  ]
 [ 5    1    1   1/2   1    1  ]
 [ 6    1    1   1/2   1    1  ]]
Lambda = 6.054173, CI= 0.010835, CR= 0.008738
Alternative weights= [0.030055 0.194962 0.161437 0.285707 0.161437 0.
 ↪166402]

Pairwise comparison of Alternatives wrt Flexibility of the space layout
[[ 1   1/4  1/5  1/9   1   1/4 ]
 [ 4    1    1    2    4    1  ]
 [ 5    1    1   1/2   5    1  ]
 [ 9   1/2   2    1    9    1  ]
 [ 1   1/4  1/5  1/9   1   1/4 ]
 [ 4    1    1    1    4    1  ]]
Lambda = 6.263322, CI= 0.052664, CR= 0.042471
Alternative weights= [0.042343 0.244746 0.191482 0.278862 0.042343 0.
 ↪200223]

Results:
  Site 1 : 0.178732
  Site 2 : 0.189963
  Site 3 : 0.139795
  Site 4 : 0.149249
  Site 5 : 0.176350
  Site 6 : 0.165911
```

```
Sorted Results:
  Site 2 : 0.189963
  Site 1 : 0.178732
  Site 5 : 0.176350
  Site 6 : 0.165911
  Site 4 : 0.149249
  Site 3 : 0.139795
```

[10]:
```
# Perform sensitivity analysis
# DrugCenter.sensit(0.4)
```

[11]:
```
# Perform Cost-effectivess and Efficient Frontier Analysis
Attributes = ['EUAC ($K)', 'Effectiveness']

EUAC = {'Site 1': 48.0,
        'Site 2': 53.3,
        'Site 3': 54.6,
        'Site 4': 60.6,
        'Site 5': 67.8,
        'Site 6': 47.5 }

EUAC_Eff = { site : (EUAC[site], site_global_wt[site]) for site in␣
 ↪alternatives }
```

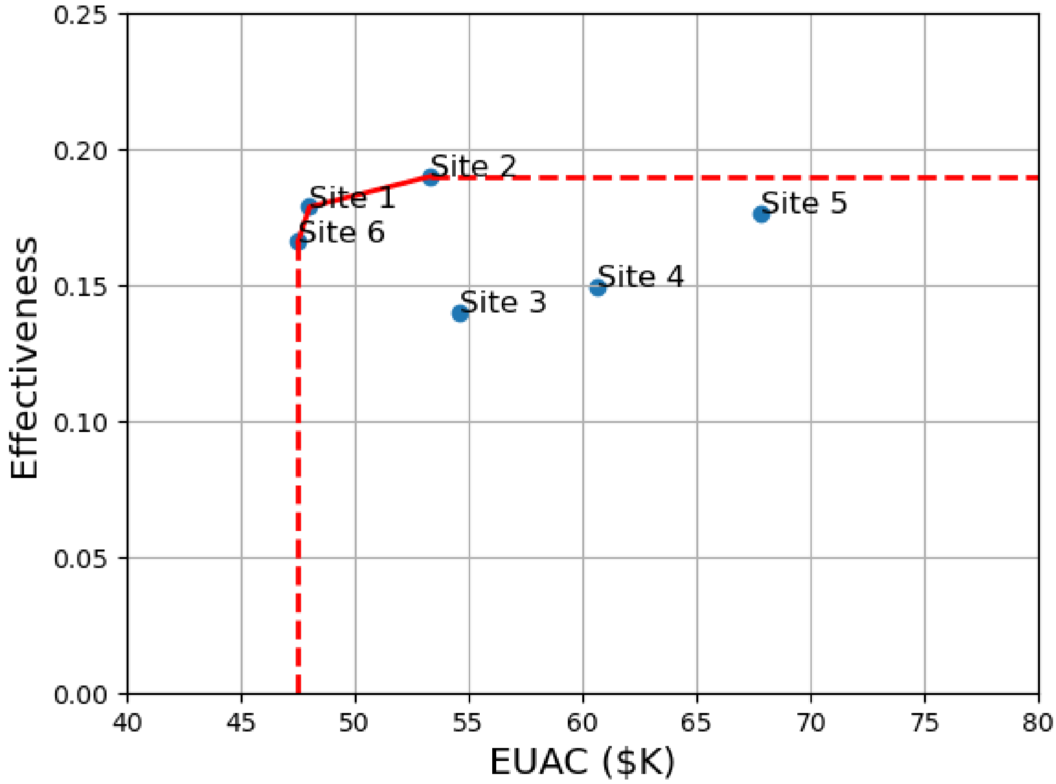
[12]:
```
# This is what EUAC_Eff is suppose to be
EUAC_Eff
```

[12]:
```
{'Site 1': (48.0, 0.17873155273071062),
 'Site 2': (53.3, 0.1899625615975271),
 'Site 3': (54.6, 0.13979537103825387),
 'Site 4': (60.6, 0.14924943329084497),
 'Site 5': (67.8, 0.17634970214294682),
 'Site 6': (47.5, 0.16591137919971657)}
```

[13]:
```
# Create a Cost-Effective Analysis Problem
reloc = Cost_Effective_Analysis(Attributes, EUAC_Eff)

# Get the efficient set
eff_set = reloc.get_efficient_set()
print("\nEfficient Sites:")
for site, values in eff_set.items():
    print(f"  {site}: {values}")
```

```
Efficient Sites:
  Site 6: (47.5, 0.16591137919971657)
  Site 1: (48.0, 0.17873155273071062)
  Site 2: (53.3, 0.1899625615975271)
```

[14]:
```
# Plot the efficient frontier
reloc.plot_efficient_frontier((40, 80), (0, 0.25), dpi=100)
```



12.3 Problem 10.4: Complete AHP and Cost Effectiveness Analysis

Source: 10_Problem_10.4_Design_Selection_Problem_complete_cost_effectivess_analysis.ipynb

[1]:
```
""" Problem 10.4: System Selection Problem:
    (a) Perform effectiveness analysis with AHP using AHP3Lmodel Class
    (b) Perform cost-effectiveness and efficient frontier analysis␣
 ↪using
        Cost_Effective_Analysis Class """
from DecisionAnalysisPy import AHP3Lmodel, Cost_Effective_Analysis
import numpy as np
```

[2]:
```
# # Effectiveness analysis using AHP model
goal = "Best System"
main_criteria =["Human Productivity", "Economics", "Design",␣
 ↪"Operations"]

# Upper triangle of criteria pairwise comparison matrix
main_cr_matrix = np.array([ 3, 3, 7,
                               2, 5,
                                  7 ])

alternatives = ["System A", "System B", "System C" ]
 # Upper triangles of alternatives pairwise comp matrix wrt each␣
 ↪criterion
alt_matrixs  = [ np.array([ 3,   5,   2 ]),
                 np.array([1/3, 1/2,  3 ]),
                 np.array([1/2, 1/7, 1/5]),
                 np.array([ 3,  1/5, 1/9])]
```

[3]:
```
# Create an instance of a 3-Level AHP model
P104 = AHP3Lmodel(goal, main_criteria, main_cr_matrix, alternatives,␣
 ↪alt_matrixs)
# Show the model
# P104.model()
```

[4]:
```
# Solve the model
sys_global_wt = P104.solve(method='Power')
# "Power", "Algebra", "RGM", "ColsNorm", "GenEigen"
```

```
Model Summary:
  Goal: Best System
  Criteria: ['Human Productivity', 'Economics', 'Design', 'Operations']
  Alternatives: ['System A', 'System B', 'System C']

Criteria w.r.t. Goal Best System:

[[ 1    3    3    7  ]
```

```
 [1/3   1    2    5  ]
 [1/3  1/2   1    7  ]
 [1/7  1/5  1/7   1  ]]
Lambda_max = 4.212088, CI= 0.070696, CR= 0.078551
Criteria Weights= [0.513052 0.246592 0.193575 0.046781]

Alternatives w.r.t. criterion Human Productivity
[[ 1    3    5  ]
 [1/3   1    2  ]
 [1/5  1/2   1  ]]
Lambda_max = 3.003695, CI= 0.001847, CR= 0.003185
Local Weights= [0.648329 0.229651 0.12202 ]

Alternatives w.r.t. criterion Economics
[[ 1   1/3  1/2 ]
 [ 3    1    3  ]
 [ 2   1/3   1  ]]
Lambda_max = 3.053622, CI= 0.026811, CR= 0.046225
Local Weights= [0.157056 0.593634 0.249311]

Alternatives w.r.t. criterion Design
[[ 1   1/2  1/7 ]
 [ 2    1   1/5 ]
 [ 7    5    1  ]]
Lambda_max = 3.014152, CI= 0.007076, CR= 0.012200
Local Weights= [0.093813 0.166593 0.739594]

Alternatives w.r.t. criterion Operations
[[ 1    3   1/5 ]
 [1/3   1   1/9 ]
 [ 5    9    1  ]]
Lambda_max = 3.029064, CI= 0.014532, CR= 0.025055
Local Weights= [0.178178 0.070418 0.751405]

Results:
  System A :0.397850
  System B :0.299750
  System C :0.302399

Sorted Results:
  System A :0.397850
  System C :0.302399
  System B :0.299750
```

[5]:

```
# Performance sensitivity analysis
# P104.sensit()
```

[6]:
```
# Perform cost-effectiveness and efficient frontier analysis
Attributes = ['EUAC ($K)', 'Effectiveness']

EUAC = {'System A' : 100,
        'System B' :  80,
        'System C' : 110 }

EUAC_Eff = { system : (EUAC[system],sys_global_wt[system])
                for system in alternatives}
EUAC_Eff
```

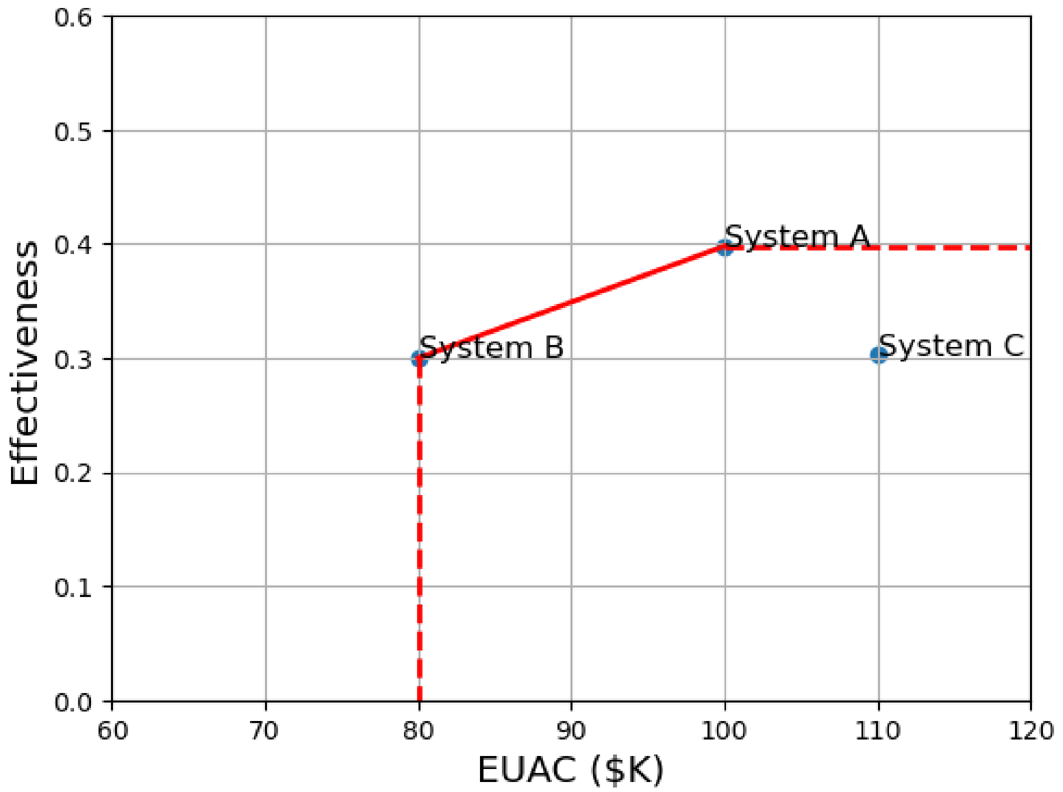
[6]:
```
{'System A': (100, 0.397850324907071445),
 'System B': (80, 0.29975039656886754),
 'System C': (110, 0.3023992785240608)}
```

[7]:
```
# Create a Cost-Effective Analysis Problem
SysSelect = Cost_Effective_Analysis(Attributes, EUAC_Eff)
```

[8]:
```
# Get the efficient set
eff_set = SysSelect.get_efficient_set()
print("\nEfficient system:")
for sys, vals in eff_set.items():
    print(f"  {sys}: {vals}")
```

```
Efficient system:
  System B: (80, 0.29975039656886754)
  System A: (100, 0.397850324907071445)
```

[9]:
```
# Plot the efficient frontier
SysSelect.plot_efficient_frontier((60, 120), (0, 0.6), dpi=100)
```


System A
System B
System C
Effectiveness
0.6
0.5
0.4
0.3
0.2
0.1
0.0
60
70
80
90
100
110
120
EUAC ($K)

```
[ ]:
```