DApy: Computational Tools for Decision Analysis in Python

User Guide

POH, Kim Leng
Department of Industrial Systems Engineering & Management
National University of Singapore

pohkimleng@nus.edu.sg

August 26, 2026

Abstract

This document describes **DApy**, 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.

Contents

1. Introduction

1.1 About DApy

DApy provides a set of computational and utility tools in Python for performing Decision Analysis tasks. It was developed as a learning tools for students reading the **IE5203 Decision Analysis** course and its variants in the Department of Industrial Systems Engineering and Management, College of Design and Engineering, National University of Singapore.

1.2 Getting Started

1.2.1 The distribution files

DApy is distributed in 3 parts:

1. DApy_user_guide.pdf (this document)
2. DApy_src.zip
3. DApy_examples.zip

DApy_src.zip contains the source code and support files for installation. It is organized as follows:

```
DApy_src/
  DApy/
    __init__.py
    risk.py
    probability.py
    bn.py
    cara.py
    mcdm.py
    utils.py
  setup.py
```

DApy_examples.zip contains example code for accessing DApy, arranged by chapters of the lecture notes.

```
DApy_examples/
  ch 4
  ch 5
  ...
  ch 9
```

```
ch 10
```

1.2.2 Installation and Use

You may access the DApy classes and functions in your Python script or Jupyter notebooks in one of the ways described here.

I. Install DApy as a Python package (best)

This is cleanest and the most flexible way to use DApy once it is properly set up. We assume that you are familiar with python environments and pip installation process.

1. Fully unpack the files in DApy_src.zip to a temporary folder (use the lecture notes password if prompted).
2. Open a command prompt/terminal/shell and change the current directory to the folder you have unpacked the source code, e.g., DApy_src.
3. Activate the Python environment you want to run DApy in.
4. run `"pip install DApy ."` to install DApy as a package. Take note of the required dot in the command line.
5. If pip reports no error, you are all set, and you may assess the DApy classes and functions using standard python import statements.

   Examples:

```
from DApy.risk import DiscreteDeal
from DApy.bn import DSepAnalysis
from DApy.cara import ExponUtilityFunction
from DApy.probability import FitPDF
from DApy.mcdm import AHPMatrix
```

6. You unpack and run the example py and ipynb files from anywhere.
7. You may delete temporary folder where you did the installation.

Note that if you are using the Anaconda distribution, you may do a pip install in a conda environment.

II. Copy the DApy folder to the current working directory

This method does not require any installation with pip. The drawback is that you have to copy the DApy folder to every directory you want to work in with your .py or .ipynb files.

1. Unpack DApy_src.zip and copy DApy folder with its contents to the same directory as your python scripts or jupyter notebooks. The directory structure may looks like:

```
myDAproject/
  mycode_1.py
  mycode_2.ipynb
  ...
  DApy/
    risk.py
    probability.py
    bn.py
    cara.py
    mcdm.py
    utils.py
```

2. You may assess the DApy classes and functions using python import statements:

   Examples:

```
from DApy.risk import DiscreteDeal
from DApy.bn import DSepAnalysis
from DApy.cara import ExponUtilityFunction
from DApy.probability import FitPDF
from DApy.mcdm import AHPMatrix
```

3. To run the examples provided in DApy_examples.zip, you will you will need to copy the DApy folder to **ALL** the unpacked chapter folders.

III. Place DApy folder in the parent directory for the work folder

This method avoids duplicating the DApy folder in every working directory. The drawback is that you will have to modify the import statements in every script or notebook.

1. Unpack and copy the DApy folder to your working directory. The structure may looks like:

```
myDAproject/
  DApy/
    risk.py
    bn.py
    bn.py
    cara.py
    mcdm.py
    utils.py
  my_assignments/
    my_code_1.py
    my_code_2.py
    my_code_3.ipynb
    ...
  examples/
    ch 4/
      party_problem_risk_profiles.py
      ...
    ch 5/
      d_seperation_example_1.py
      ...
```

2. You may assess the DApy classes and functions by adding the following code in the header to modify the search path:

```
## Add the parent directory '..' to the search path
import sys, os
sys.path.append(os.path.abspath('..'))
## Import what you want
from DApy.risk import DiscreteDeal
from DApy.bn import DSepAnalysis
from DApy.cara import ExponUtilityFunction
```

IV. Set up as an IDE project

This method manages all the files as a project within the IDE, such as for example, Spyder, PyCharm, VS Code.

1. Create a new project in your IDE.
2. Set the project root directory.
3. Unpack and copy the DApy folder to the project or ensure its path is included. See your IDE documentations for details.

V. Set the PYTHONPATH environment variable

In this method, instead of adding code to your script, you set the PYTHONPATH environment variable in the operating system. This is cleaner than modifying the sys.path variable in the code, and is suitable for advanced users.

1. **Windows:**

```
set PYTHONPATH=C:\path\to\DApy_parent
```

2. **Linux/Mac:**

```
export PYTHONPATH=/path/to/DApy_parent
```

2. The Party Problem

2.1 Plotting Risk Profiles

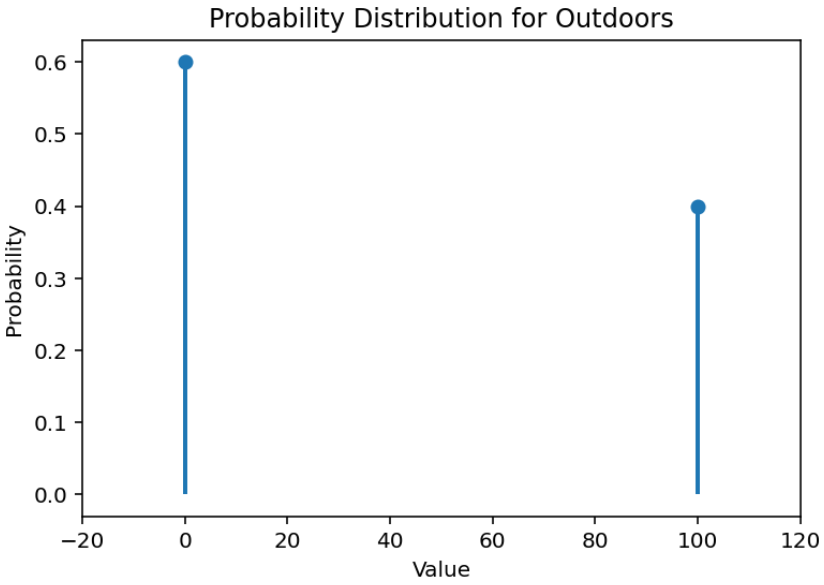
2.1.1 Plotting Individual Risk Profiles

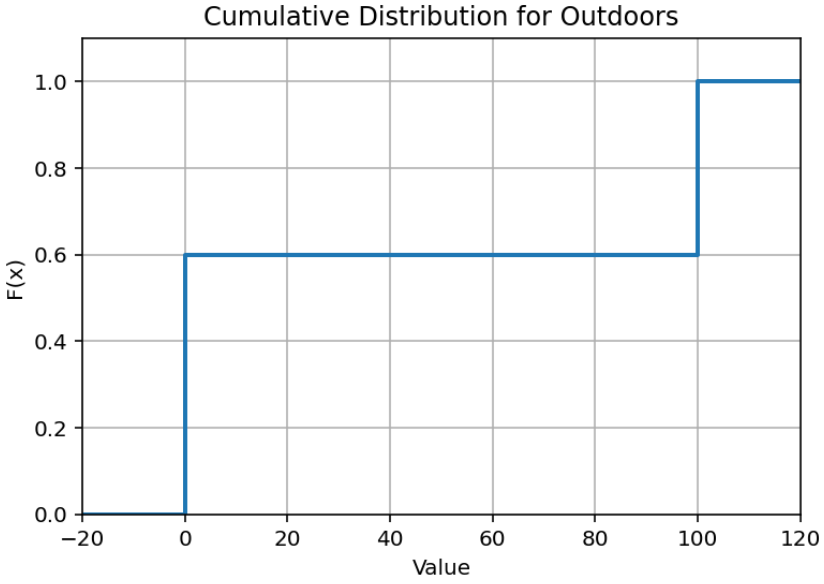
We plot the risk profiles for the Party Problem in Chapter 4 using the DiscreteDeal class.

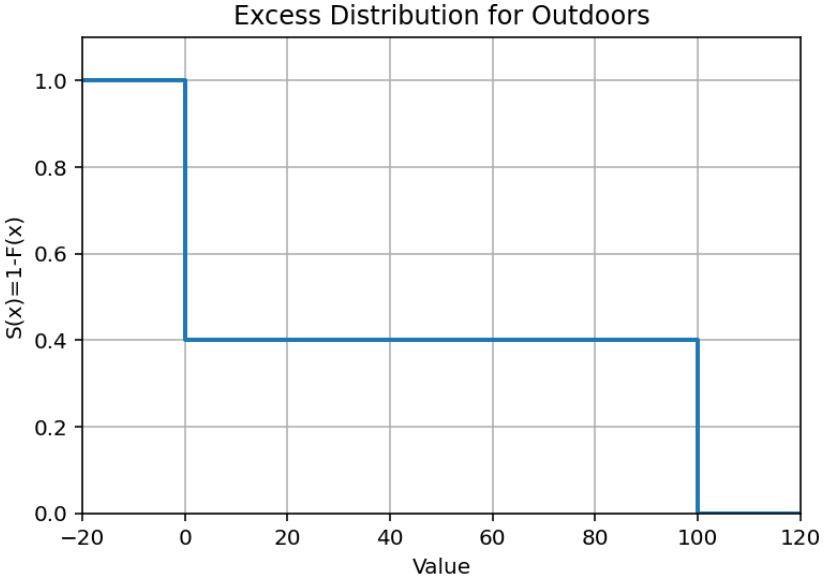
```
from  DApy.risk import DiscreteDeal
import numpy as np
# Party Problem - Risk Profiles plotting

# Create the 3 deals
Outdoors = DiscreteDeal(
    name= 'Outdoors',
    x= np.array( [0, 100]),
    p= np.array([0.6, 0.4])
    )
Porch = DiscreteDeal(
    name='Porch',
    x= np.array( [20, 90]),
    p= np.array([0.6, 0.4])
    )
Indoors = DiscreteDeal(
    name= 'Indoors',
    x= np.array([40, 50]),
    p= np.array([0.4, 0.6])
    )
```

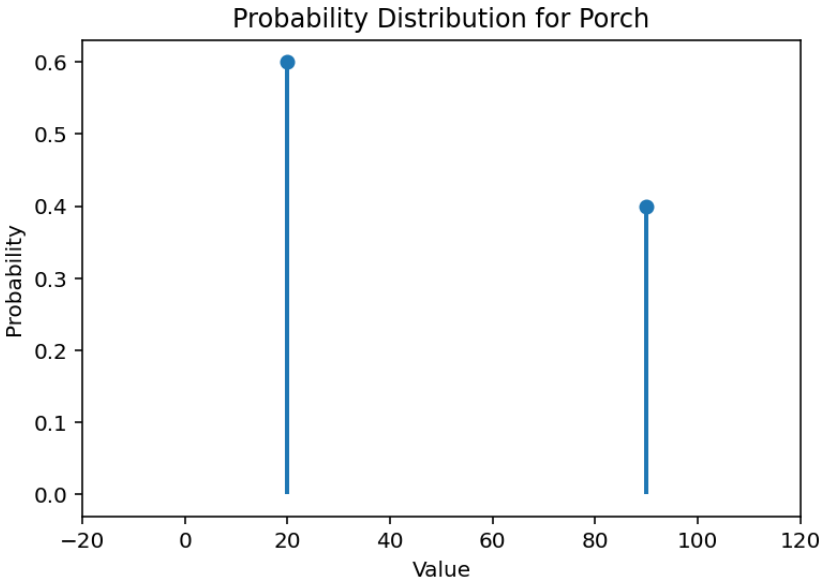
You can plot all the risk profiles individually by specifying the outcome value range.

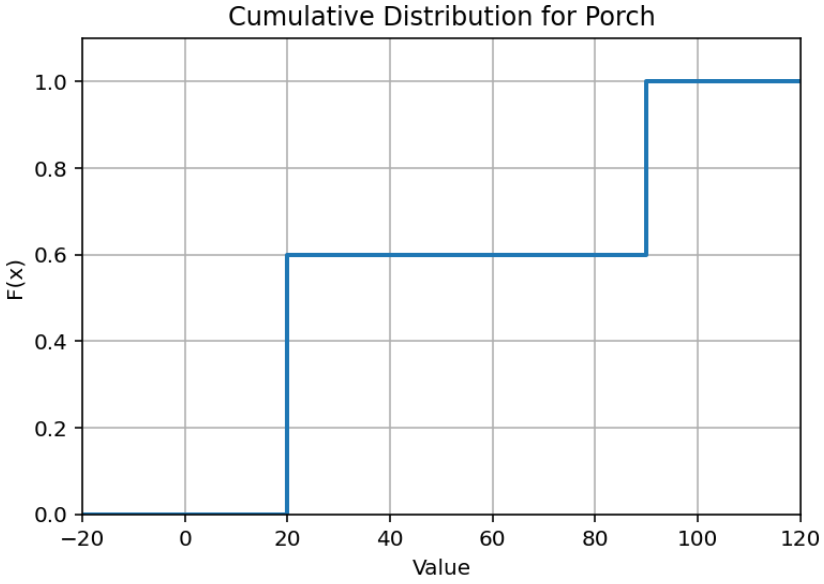
```
# Plot all risk profiles with same x range
xlim=(-20, 120) # x limits for plotting risk profiles
for deal in [Outdoors, Porch, Indoors]:
    deal.plot_pmf(xlim=xlim)
    deal.plot_cdf(xlim=xlim)
    deal.plot_epf(xlim=xlim)
```

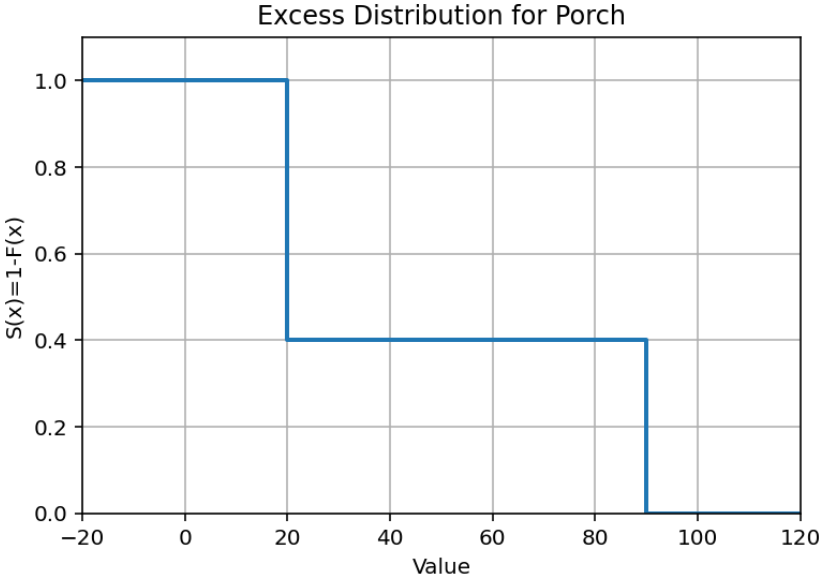

Probability Distribution for Outdoors
Probability
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−20
0
20
40
60
80
100
120
Value

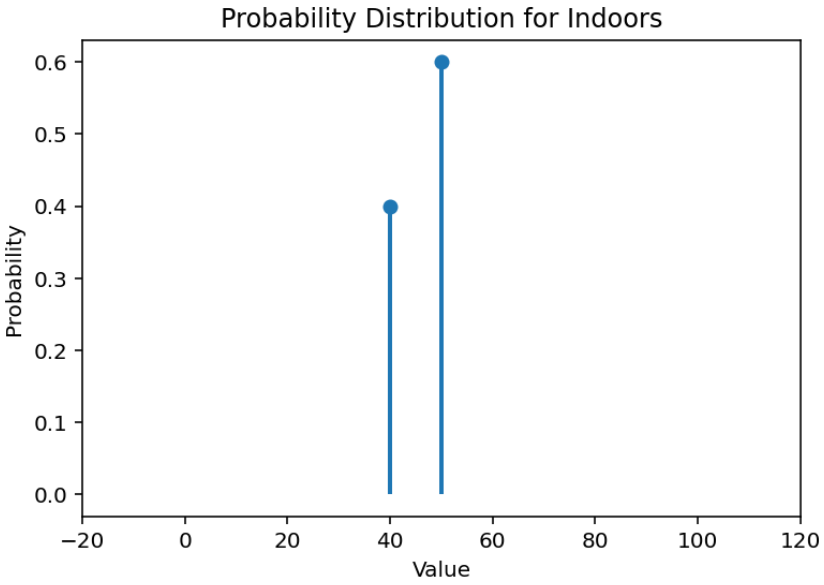

Cumulative Distribution for Outdoors
F(x)
0.0
0.2
0.4
0.6
0.8
1.0
−20
0
20
40
60
80
100
120
Value

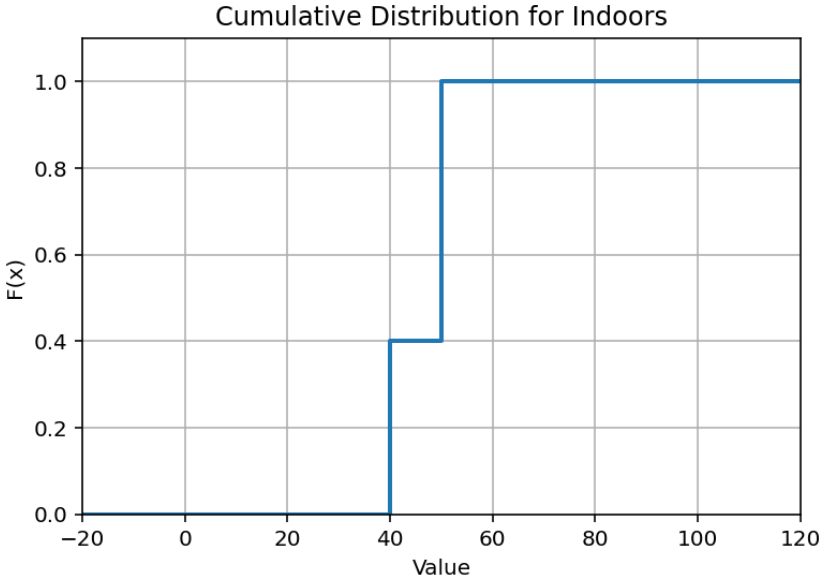

Excess Distribution for Outdoors
S(x)=1-F(x)
0.0
0.2
0.4
0.6
0.8
1.0
−20
0
20
40
60
80
100
120
Value

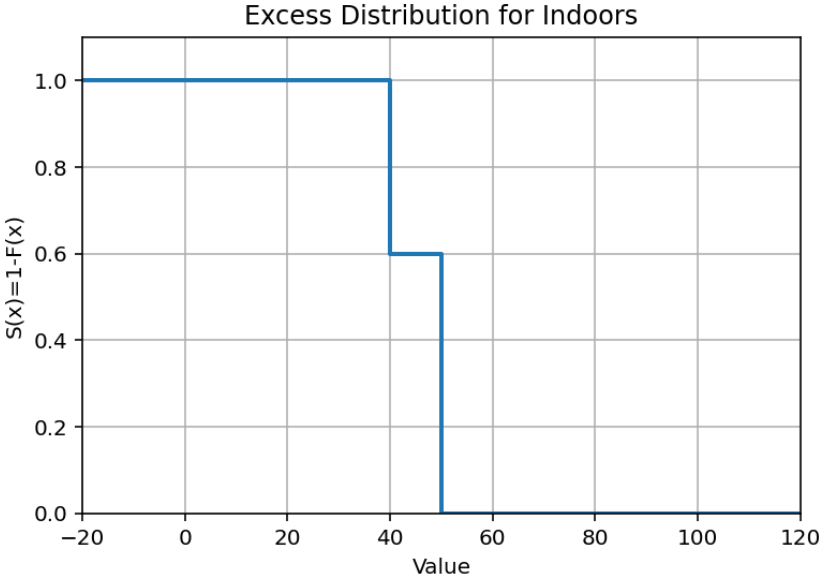

Probability Distribution for Porch
Probability
Value
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−20
0
20
40
60
80
100
120


Cumulative Distribution for Porch
F(x)
Value
0.0
0.2
0.4
0.6
0.8
1.0
−20
0
20
40
60
80
100
120


Excess Distribution for Porch
S(x)=1-F(x)
Value
0.0
0.2
0.4
0.6
0.8
1.0
−20
0
20
40
60
80
100
120


Probability Distribution for Indoors
Probability
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−20
0
20
40
60
80
100
120
Value

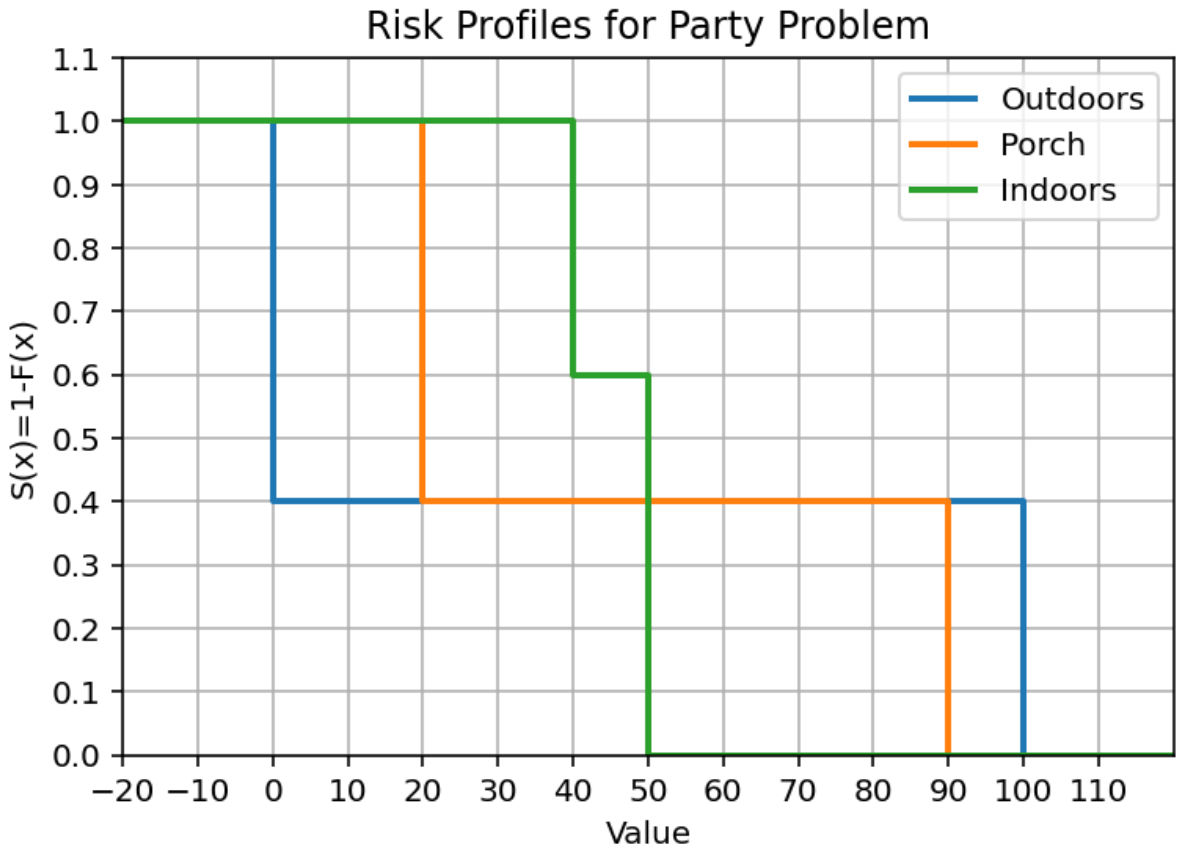

Cumulative Distribution for Indoors
F(x)
0.0
0.2
0.4
0.6
0.8
1.0
−20
0
20
40
60
80
100
120
Value


Excess Distribution for Indoors
S(x)=1-F(x)
0.0
0.2
0.4
0.6
0.8
1.0
−20
0
20
40
60
80
100
120
Value

2.1.2 Comparing Risk Profiles Graphically

We can compare the EPFs (CDFs) of all the deals on the same plot. First order stochastic dominance may be identified visually immediately. The combined plot also helps determine which are the pairs of deals to compare for second-order stochastic dominance analysis.

```
# Compare the EPF and customize the plot.
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
for deal in [Outdoors, Porch, Indoors]:
    ax = deal.plot_epf(xlim=xlim, ax=ax)
ax.set_title('Risk Profiles for Party Problem')
ax.set_yticks(np.arange(0, 1.2, 0.1))
ax.set_xticks(np.arange(-20, 120, 10))
ax.legend()
plt.show()
```



2.2 Stochastic Dominance Analysis

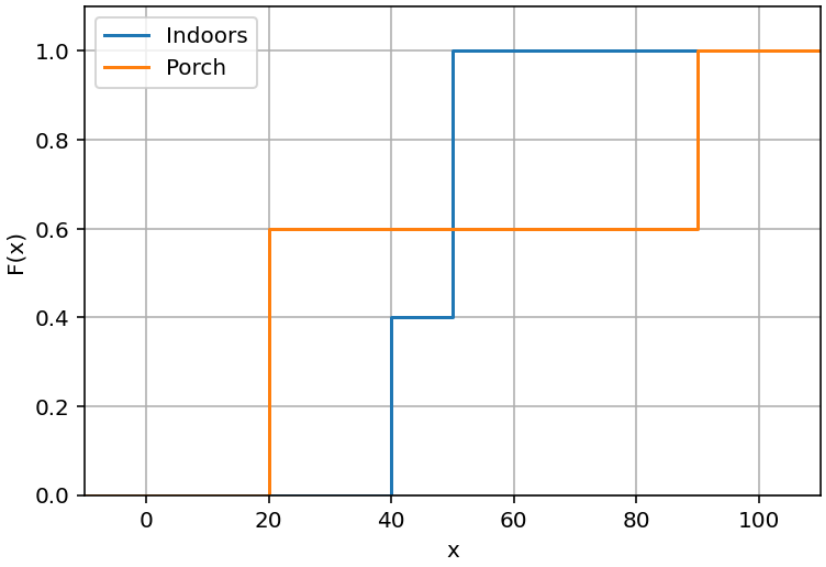
2.2.1 First-Order Stochastic Dominance Analysis

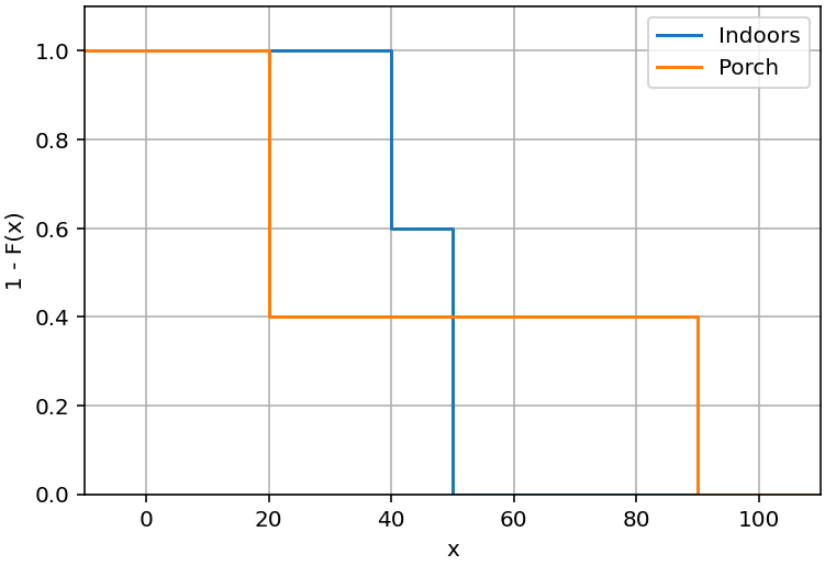
In the previous example, we observed graphically that there are no first-order stochastic dominance among all three deals. Here, we will do a numerical check for illustration purposes.

To determine if Indoors 1st order dominates Porch.

```
DiscreteDeal.first_order_SD(Indoors, Porch, xlim=xlim)
```

```
Indoors does not 1SD Porch
```

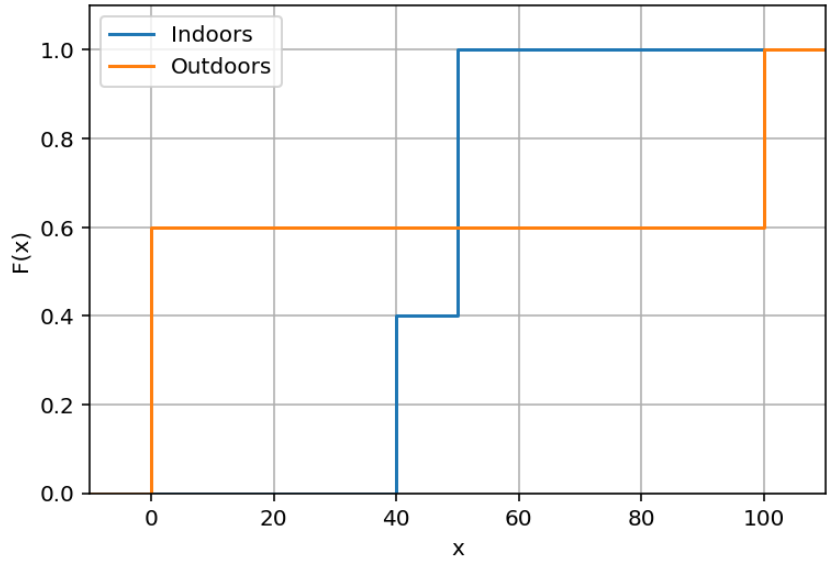


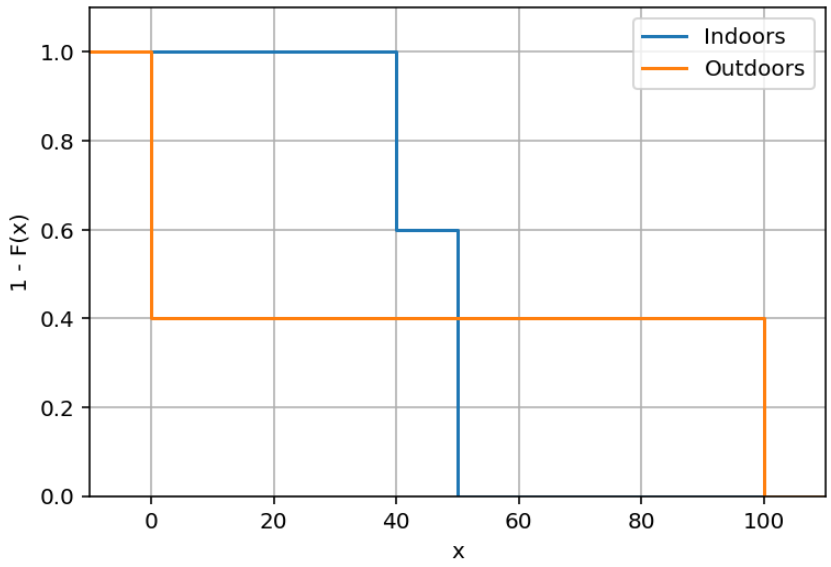


To determine if Indoors 1st order dominates Outdoors.

```
DiscreteDeal.first_order_SD(Indoors, Outdoors, xlim=xlim)
```

```
Indoors does not 1SD Outdoors
```

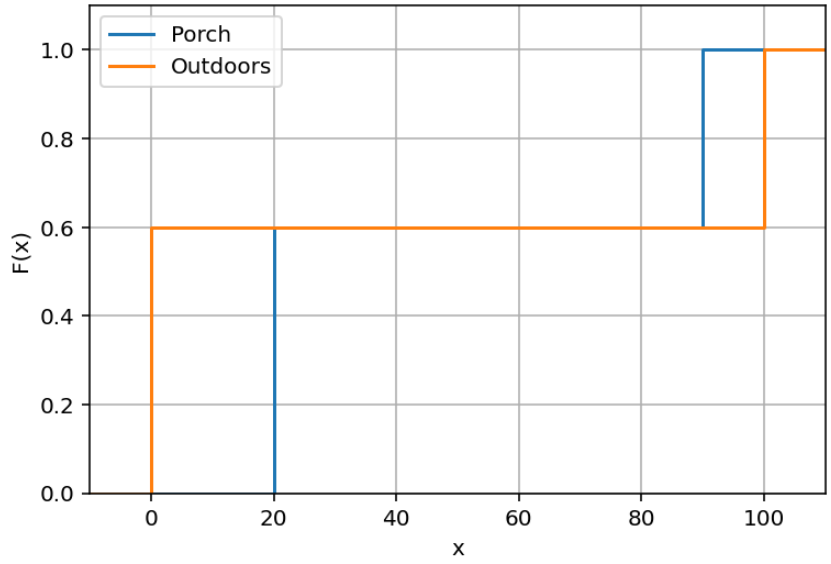


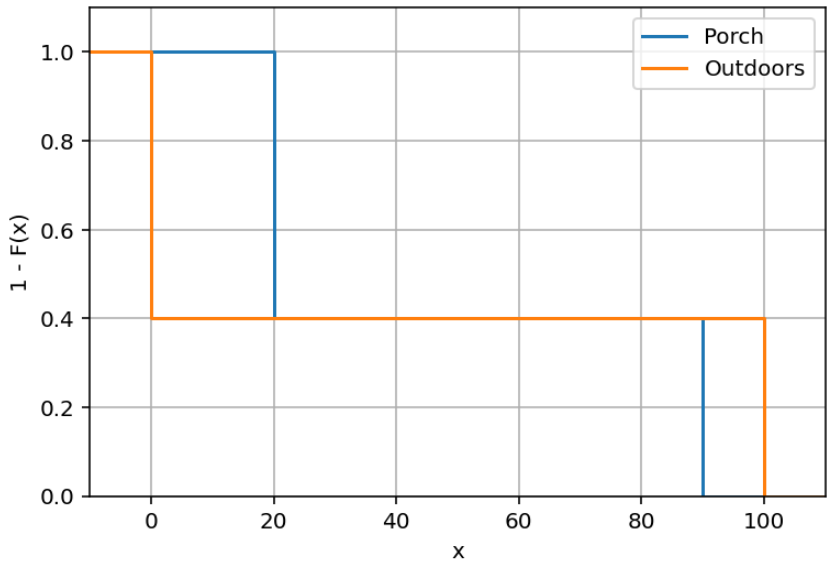


To determine if Indoors 1st order dominates Outdoors.

```
DiscreteDeal.first_order_SD(Porch, Outdoors, xlim=xlim)
```

```
Porch does not 1SD Outdoors
```





2.2.2 Second-Order Stochastic Dominance Analysis

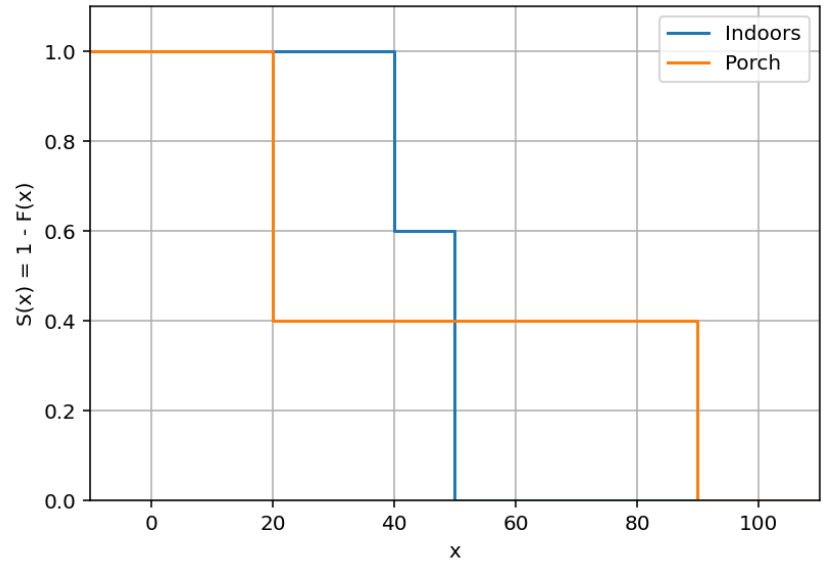
From the risk profiles plotted above, we conclude that we only need to check the following:

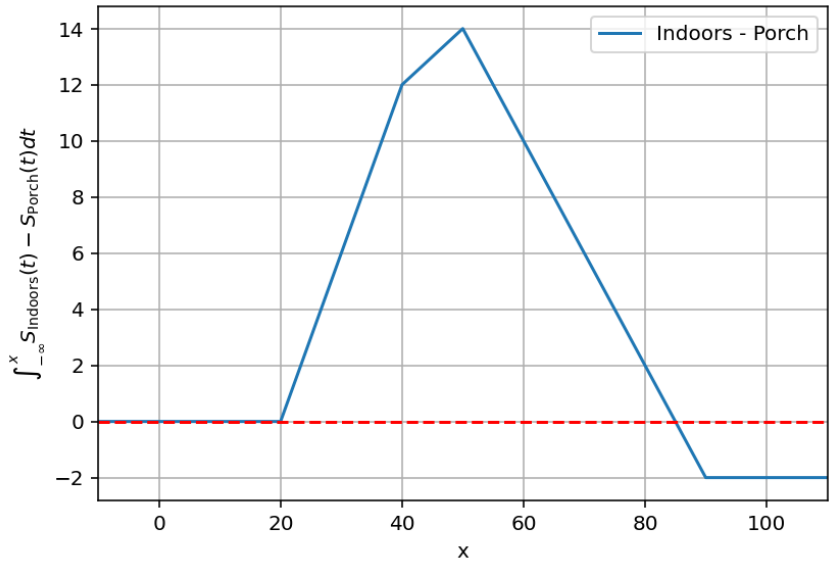
1. If Indoors 2SD Porch
2. If Indoors 2SD Outdoors
3. If Porch 2SD. Outdoors

To determine if Indoors 2nd order dominates Porch.

```
DiscreteDeal.second_order_SD(Indoors, Porch, xlim=xlim, plot=True)
```

```
Indoors does not 2SD Porch
```

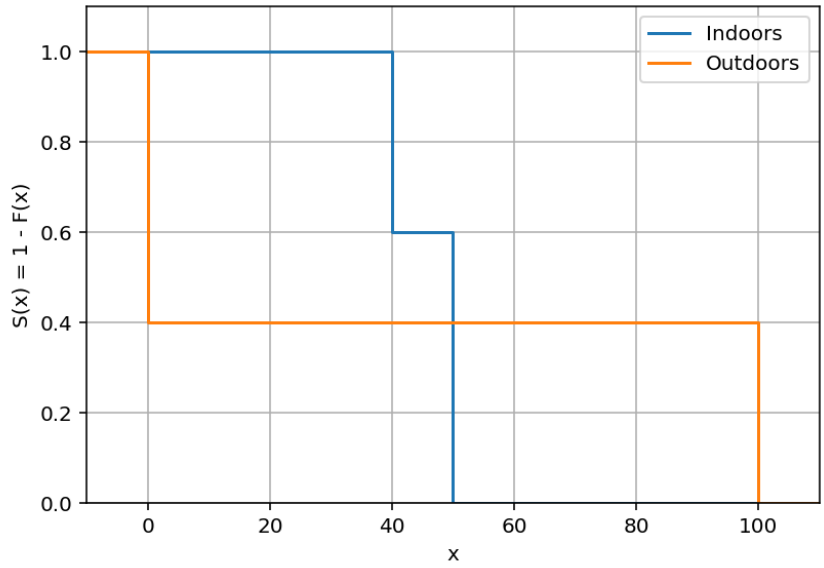


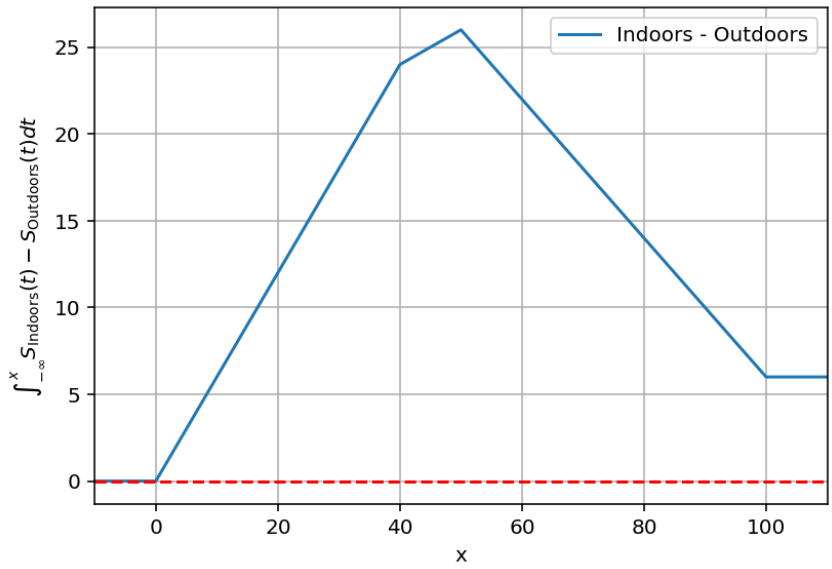


to determine if Indoors 2nd order dominates Outdoors.

```
DiscreteDeal.second_order_SD(Indoors, Outdoors, xlim=xlim, plot=True)
```

```
Indoors 2SD Outdoors
```

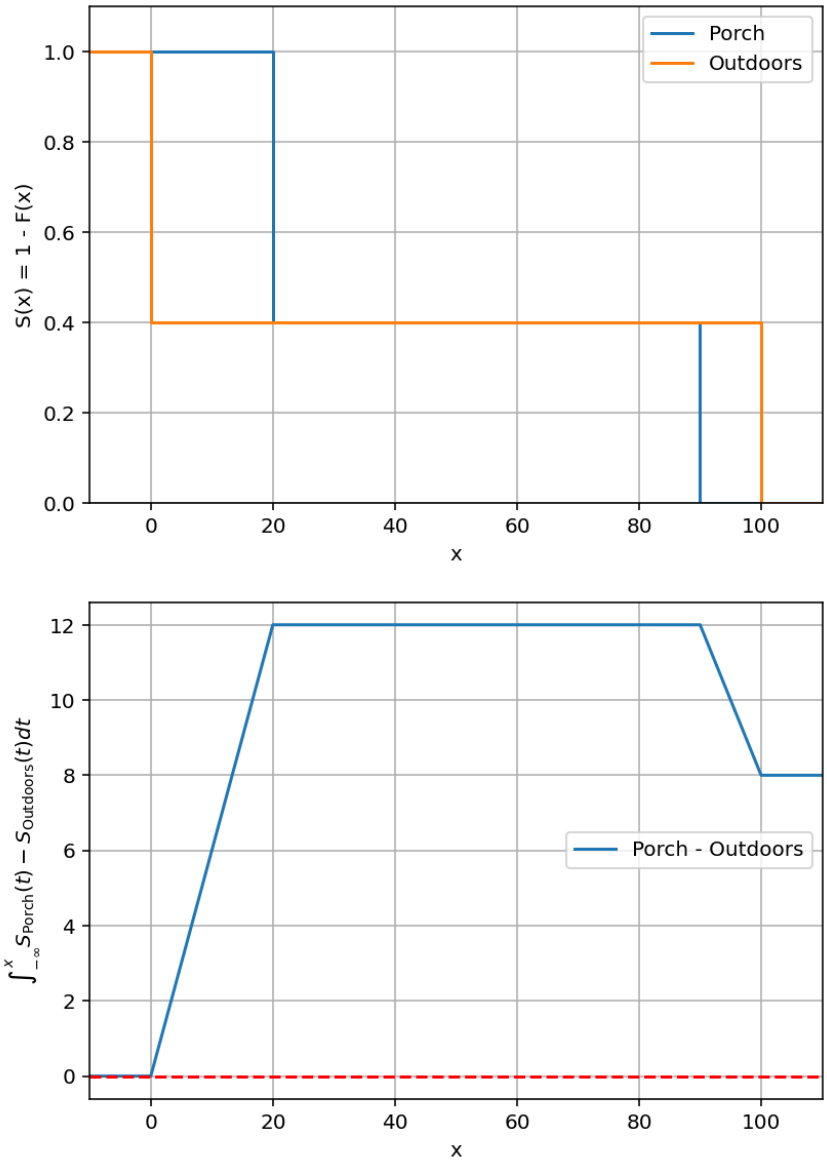




to determine if Porch 2nd order dominates Outdoors

```
DiscreteDeal.second_order_SD(Porch, Outdoors, xlim=xlim, plot=True)
```

```
Porch 2SD Outdoors
```



3. Bayesian Networks

3.1 d-Seperation Analysis I

We first set up the DAG representing the graphical structure of the Bayesian network, using the networkx package.

```
import networkx as nx
from DApy.bn import DSepAnalysis

# Define the nodes and their parents
Parents = {'A': [],
           'B': ['D'],
           'C': [],
           'D': ['A'],
           'E': ['B','C'],
           'F': ['E'],
           'G': [],
           'H': ['D','G'],
           'I': ['E'],
           'J': ['F'],
           'K': ['H'],
           'L': ['H','I'],
           'M': ['I']  }

# Generate the list directed arcs
Edges = [(p, child) for child, par in Parents.items() for p in par]

# Create the graph
G = nx.DiGraph()
G.add_edges_from(Edges)
```

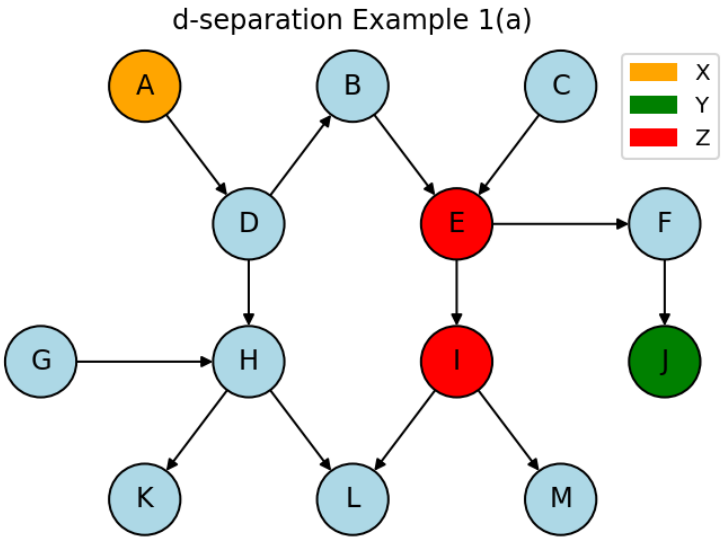
We define the positions to draw the nodes.

```
# Spring layout will be used if positions are not provide.
sin60 = 3**(1/2)/2
pos = {'K': (0, 0),
       'L': (1, 0),
       'M': (2, 0),
       'G': (-0.5, sin60),
       'H': ( 0.5, sin60),
       'I': ( 1.5, sin60),
       'J': ( 2.5, sin60),
       'D': ( 0.5, 2*sin60),
       'E': ( 1.5, 2*sin60),
       'F': ( 2.5, 2*sin60),
       'A': (0, 3*sin60),
       'B': (1, 3*sin60),
       'C': (2, 3*sin60)}
```

Example 1(a)

Let $X = \{A\}, Y = \{J\}, Z = \{E, I\}$.

```
dag1 = DSepAnalysis(G)
X = {'A'}
Y = {'J'}
Z = {'E','I'}

dag1.plot(X,Y,Z, pos=pos, title="d-separation Example 1(a)")
```



Use networkx built-in d-separation analyser to determine if $\{A\} \perp \{J\} \quad | \quad \{E, I\}$ is True or False.

```
print("\nUsing networkx built-in d-separation analyser")
print(f"  {X}⊥{Y}|{Z} is {dag1.is_d_separated(X, Y, Z)}")
```

```
Using networkx built-in d-separation analyser

   {'A'} ⊥ {'J'}|{'E','I'} is True
```

Networkx package only gives you True or False; there is no details.

DSepAnalysis class can do path-by-path block/unblock analysis between X and Y, and determine if the d-seperation is satisfied.

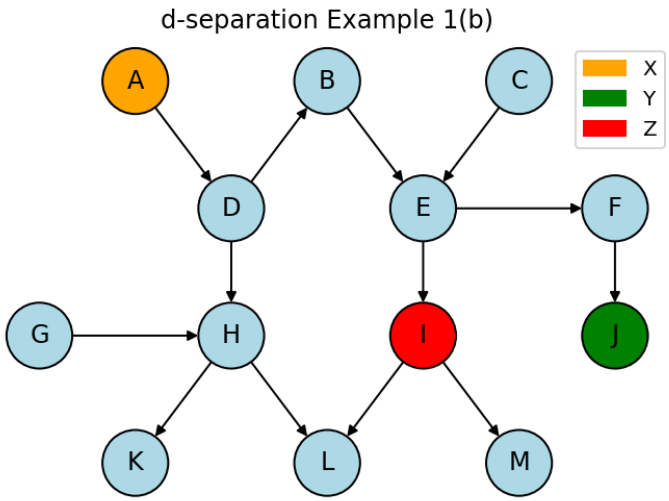
```
# Path-by-path analysis.
dsep, results = dag1.path_blocking_analysis(X, Y, Z)
```

```
Path-by-Path Analysis:
============================================================
Path: A - D - B - E - F - J
------------------------------------------------------------
D: head-to-tail, node not in Z -> Not blocking
B: head-to-tail, node not in Z -> Not blocking
E: head-to-tail, node in Z -> Blocking
F: head-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: A - D - H - L - I - E - F - J
------------------------------------------------------------
D: head-to-tail, node not in Z -> Not blocking
H: head-to-tail, node not in Z -> Not blocking
L: head-to-head, node not in Z -> Blocking
I: head-to-tail, node in Z -> Blocking
E: tail-to-tail, node in Z -> Blocking
F: head-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

Conclusion: {'A'}⍰{'J'}|{'E', 'I'} is True
```

Example 1(b)

Let $X = \{A\}, Y = \{J\}, Z = \{I\}$.

```
dag2 = DSepAnalysis(G)
X = {'A'}
Y = {'J'}
Z = {'I'}

dag2.plot(X,Y,Z, pos=pos, title="d-separation Example 1(b)")
```



Use networkx built-in d-separation analyser to determine if $\{A\} \perp \{J\} \mid \{I\}$ is True or False.

```
print("\nUsing networkx built-in d-separation analyser")
print(f"  {X}⊥{Y}|{Z} is {dag1.is_d_separated(X, Y, Z)}")
```

```
Using networkx built-in d-separation analyser

    {'A'} ⊥ {'J'}|{'I'} is False
```

Do path-by-path blocking analysis.

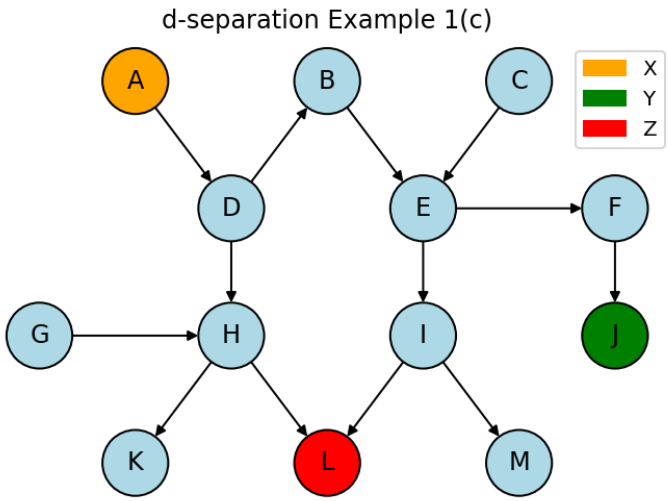
```
# Path-by-path analysis
dsep, results = dag2.path_blocking_analysis(X, Y, Z)
```

```
Path-by-Path Analysis:
============================================================
Path: A - D - B - E - F - J
------------------------------------------------------------
D: head-to-tail, node not in Z -> Not blocking
B: head-to-tail, node not in Z -> Not blocking
E: head-to-tail, node not in Z -> Not blocking
F: head-to-tail, node not in Z -> Not blocking

Path is NOT BLOCKED
============================================================

============================================================
Path: A - D - H - L - I - E - F - J
------------------------------------------------------------
D: head-to-tail, node not in Z -> Not blocking
H: head-to-tail, node not in Z -> Not blocking
L: head-to-head, node not in Z -> Blocking
I: head-to-tail, node in Z -> Blocking
E: tail-to-tail, node not in Z -> Not blocking
F: head-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

Conclusion: {'A'}⍰{'J'}|{'I'} is False
```

Example 1(c)

Let $X = \{A\}, Y = \{J\}, Z = \{L\}$.

```
dag3 = DSepAnalysis(G)
X = {'A'}
Y = {'J'}
Z = {'L'}

dag3.plot(X,Y,Z, pos=pos, title="d-separation Example 1(c)")
```




```
# Use networkx built-in d-separation analyser
print("\nUsing networkx built-in d-separation analyser")
print(f"  {X}⊥{Y}|{Z} is {dag3.is_d_separated(X, Y, Z)}")
```

```
Using networkx built-in d-separation analyser
    {'A'} ⊥ {'J'}|{'L'} is False
```

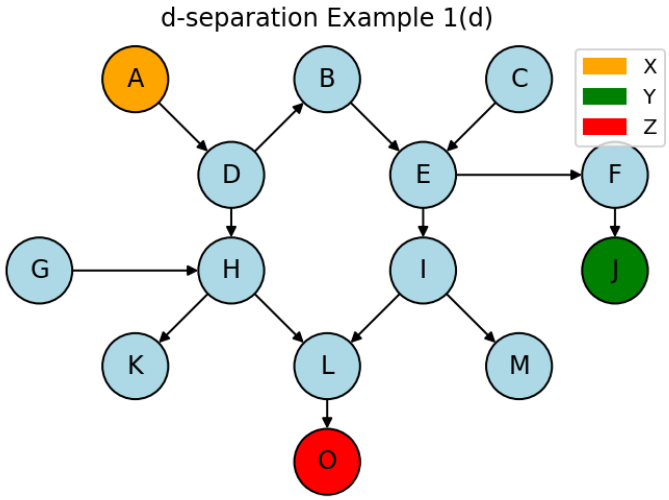
```
# Path-by-path blocking analysis
dsep, results = dag3.path_blocking_analysis(X, Y, Z)
```

```
Path-by-Path Analysis:
============================================================
Path: A - D - B - E - F - J
------------------------------------------------------------
D: head-to-tail, node not in Z -> Not blocking
B: head-to-tail, node not in Z -> Not blocking
E: head-to-tail, node not in Z -> Not blocking
F: head-to-tail, node not in Z -> Not blocking

Path is NOT BLOCKED
============================================================

============================================================
Path: A - D - H - L - I - E - F - J
------------------------------------------------------------
D: head-to-tail, node not in Z -> Not blocking
H: head-to-tail, node not in Z -> Not blocking
L: head-to-head, node in Z -> Not blocking
I: head-to-tail, node not in Z -> Not blocking
E: tail-to-tail, node not in Z -> Not blocking
F: head-to-tail, node not in Z -> Not blocking

Path is NOT BLOCKED
============================================================

Conclusion: {'A'}□{'J'}|{'L'} is False
```

Example 1(d)

Let $X = \{A\}, Y = \{J\}, Z = \{O\}$.

```
# Add arc (L)->(O) to DAG
Edges = Edges+[('L','O')]
G2 = nx.DiGraph()
G2.add_edges_from(Edges)

dag4 = DSepAnalysis(G2)
pos['O'] = (1,-1*sin60)
X = {'A'}
Y = {'J'}
Z = {'O'}

dag4.plot(X,Y,Z, pos=pos, title="d-separation Example 1(d)")
```



Use nextworkx built-in analyser.

```
print("\nUsing networkx built-in d-separation analyser")
print(f"  {X}⊥{Y}|{Z} is {dag4.is_d_separated(X, Y, Z)}")
```

```
Using networkx built-in d-separation analyser

    {'A'} ⊥ {'J'}|{'O'} is False
```

Do path-by-path analysis.

```
# Path-by-path blocking analysis
dsep, results = dag4.path_blocking_analysis(X, Y, Z)
```

```
Path-by-Path Analysis:
============================================================
Path: A - D - B - E - F - J
------------------------------------------------------------
D: head-to-tail, node not in Z -> Not blocking
B: head-to-tail, node not in Z -> Not blocking
E: head-to-tail, node not in Z -> Not blocking
F: head-to-tail, node not in Z -> Not blocking

Path is NOT BLOCKED
============================================================

============================================================
Path: A - D - H - L - I - E - F - J
------------------------------------------------------------
D: head-to-tail, node not in Z -> Not blocking
H: head-to-tail, node not in Z -> Not blocking
L: head-to-head, node not in Z but descendant ['O'] in Z -> Not blocking
I: head-to-tail, node not in Z -> Not blocking
E: tail-to-tail, node not in Z -> Not blocking
F: head-to-tail, node not in Z -> Not blocking

Path is NOT BLOCKED
============================================================

Conclusion: {'A'}⍰{'J'}|{'O'} is False
```

3.2 d-Seperation Analysis II

We first set up the DAG representing the graphical structure of the Bayesian network, using the networkx package.

```
import networkx as nx
from DApy.bn import DSepAnalysis

# Define the nodes and their parents
Parents = {'A': [],
           'B': ['C','E'],
           'C': ['A'],
           'D': ['A'],
           'E': ['C'],
           'F': ['E','G'],
           'G': ['C'],
           'H': ['F'],
           'I': ['G'] }

# List of directed arcs
Edges = [(p, child) for child, par in Parents.items() for p in par]

# Create the graph
G = nx.DiGraph()
G.add_edges_from(Edges)
```

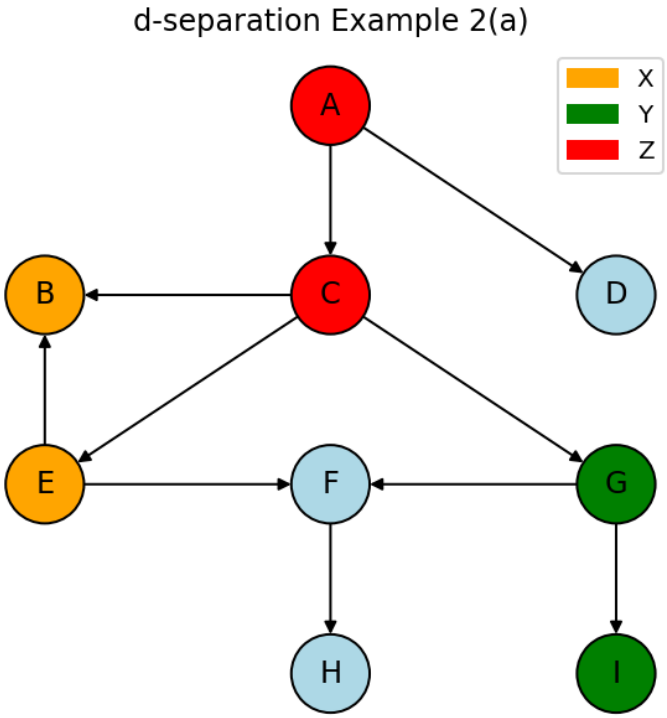
Define the positions to draw the graph.

```
# Positions to draw the nodes
# Spring layout will be used if this is not provide.
sin60 = 3**(0.5)/2
pos = {'H': (1, 0),
       'I': (2, 0),
       'E': (0, sin60),
       'F': (1, sin60),
       'G': (2, sin60),
       'B': (0, 2*sin60),
       'C': (1, 2*sin60),
       'D': (2, 2*sin60),
       'A': (1, 3*sin60) }

dag = DSepAnalysis(G)
```

Example 2(a)

Let $X = \{B, E\}, Y = \{G, I\}, Z = \{A, C\}$

```
X = {'B','E'}
Y = {'G','I'}
Z = {'A','C'}

dag.plot(X, Y, Z, pos=pos, title="d-separation Example 2(a)",
                          figsize=(5,5))
```



Use networkx built-in d-separation analyser.

```
print("\nUsing networkx built-in d-separation analyser")
print(f"  {X}⊥{Y}|{Z} is {dag.is_d_separated(X, Y, Z)}")
```

```
Using networkx built-in d-separation analyser

    {'B','E'} ⊥ {'G','I'}|{'A','C'} is True
```

Perform path-by-path blocking analysis.

```
# Path-by-path blocking analysis
dsep, results = dag.path_blocking_analysis(X, Y, Z)
```

```
Path-by-Path Analysis:
============================================================
Path: B - C - E - F - G
------------------------------------------------------------
C: tail-to-tail, node in Z -> Blocking
E: head-to-tail, node not in Z -> Not blocking
F: head-to-head, node not in Z -> Blocking

Path is BLOCKED
============================================================

============================================================
Path: B - C - G
------------------------------------------------------------
C: tail-to-tail, node in Z -> Blocking

Path is BLOCKED
============================================================

============================================================
Path: B - E - C - G
------------------------------------------------------------
E: head-to-tail, node not in Z -> Not blocking
C: tail-to-tail, node in Z -> Blocking

Path is BLOCKED
============================================================

============================================================
Path: B - E - F - G
------------------------------------------------------------
E: tail-to-tail, node not in Z -> Not blocking
F: head-to-head, node not in Z -> Blocking

Path is BLOCKED
============================================================

============================================================
Path: B - C - E - F - G - I
------------------------------------------------------------
C: tail-to-tail, node in Z -> Blocking
E: head-to-tail, node not in Z -> Not blocking
F: head-to-head, node not in Z -> Blocking
```

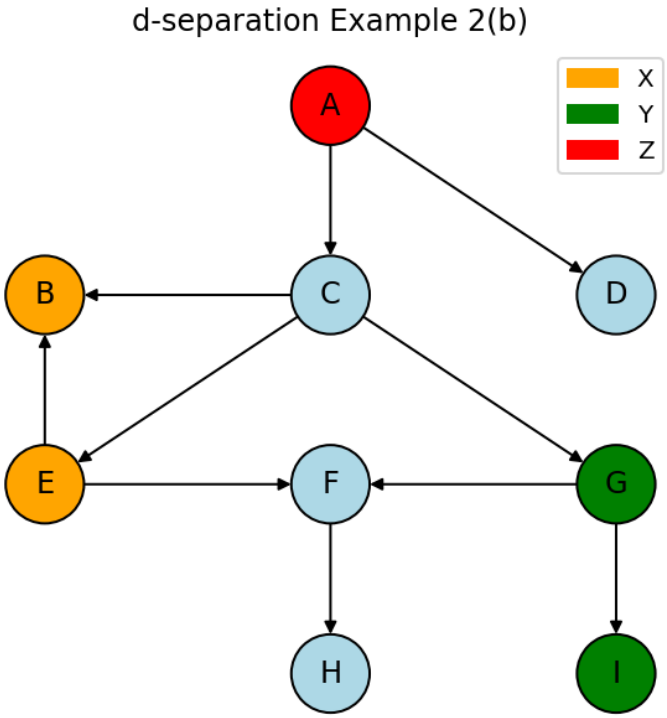
```
G: tail-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: B - C - G - I
------------------------------------------------------------
C: tail-to-tail, node in Z -> Blocking
G: head-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: B - E - C - G - I
------------------------------------------------------------
E: head-to-tail, node not in Z -> Not blocking
C: tail-to-tail, node in Z -> Blocking
G: head-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: B - E - F - G - I
------------------------------------------------------------
E: tail-to-tail, node not in Z -> Not blocking
F: head-to-head, node not in Z -> Blocking
G: tail-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: E - C - G
------------------------------------------------------------
C: tail-to-tail, node in Z -> Blocking

Path is BLOCKED
============================================================

============================================================
Path: E - B - C - G
------------------------------------------------------------
B: head-to-head, node not in Z -> Blocking
C: tail-to-tail, node in Z -> Blocking

Path is BLOCKED
============================================================
```

```
============================================================
Path: E - F - G
------------------------------------------------------------
F: head-to-head, node not in Z -> Blocking

Path is BLOCKED
============================================================

============================================================
Path: E - C - G - I
------------------------------------------------------------
C: tail-to-tail, node in Z -> Blocking
G: head-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: E - B - C - G - I
------------------------------------------------------------
B: head-to-head, node not in Z -> Blocking
C: tail-to-tail, node in Z -> Blocking
G: head-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: E - F - G - I
------------------------------------------------------------
F: head-to-head, node not in Z -> Blocking
G: tail-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

Conclusion: {'B','E'}⍰{'G','I'}|{'A','C'} is True
```

Example 2(b)

Let $X = \{B, E\}, Y = \{G, I\}, Z = \{A\}$

```
X = {'B','E'}
Y = {'G','I'}
Z = {'A'}

dag.plot(X, Y, Z, pos=pos, title="d-separation Example 2(b)",
                          figsize=(5,5))
```



Use networkx built-in d-separation analyser.

```
print("\nUsing networkx built-in d-separation analyser")
print(f"  {X}⊥{Y}|{Z} is {dag.is_d_separated(X, Y, Z)}")
```

```
Using networkx built-in d-separation analyser

    {'B','E'} ⊥ {'G','I'}|{'A'} is False
```

Do path-by-path analysis.

```
# Path-by-path blocking analysis
dsep, results = dag.path_blocking_analysis(X, Y, Z)
```

```
Path-by-Path Analysis:
============================================================
Path: B - C - E - F - G
------------------------------------------------------------
C: tail-to-tail, node not in Z -> Not blocking
E: head-to-tail, node not in Z -> Not blocking
F: head-to-head, node not in Z -> Blocking

Path is BLOCKED
============================================================

============================================================
Path: B - C - G
------------------------------------------------------------
C: tail-to-tail, node not in Z -> Not blocking

Path is NOT BLOCKED
============================================================

============================================================
Path: B - E - C - G
------------------------------------------------------------
E: head-to-tail, node not in Z -> Not blocking
C: tail-to-tail, node not in Z -> Not blocking

Path is NOT BLOCKED
============================================================

============================================================
Path: B - E - F - G
------------------------------------------------------------
E: tail-to-tail, node not in Z -> Not blocking
F: head-to-head, node not in Z -> Blocking

Path is BLOCKED
============================================================

============================================================
Path: B - C - E - F - G - I
------------------------------------------------------------
C: tail-to-tail, node not in Z -> Not blocking
E: head-to-tail, node not in Z -> Not blocking
F: head-to-head, node not in Z -> Blocking
```

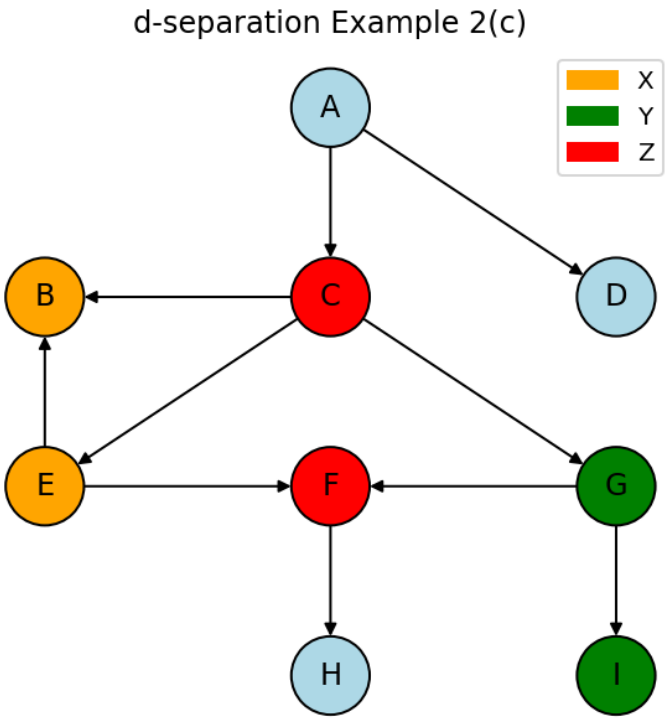
```
G: tail-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: B - C - G - I
------------------------------------------------------------
C: tail-to-tail, node not in Z -> Not blocking
G: head-to-tail, node not in Z -> Not blocking

Path is NOT BLOCKED
============================================================

============================================================
Path: B - E - C - G - I
------------------------------------------------------------
E: head-to-tail, node not in Z -> Not blocking
C: tail-to-tail, node not in Z -> Not blocking
G: head-to-tail, node not in Z -> Not blocking

Path is NOT BLOCKED
============================================================

============================================================
Path: B - E - F - G - I
------------------------------------------------------------
E: tail-to-tail, node not in Z -> Not blocking
F: head-to-head, node not in Z -> Blocking
G: tail-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: E - C - G
------------------------------------------------------------
C: tail-to-tail, node not in Z -> Not blocking

Path is NOT BLOCKED
============================================================

============================================================
Path: E - B - C - G
------------------------------------------------------------
B: head-to-head, node not in Z -> Blocking
C: tail-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================
```

```
============================================================
Path: E - F - G
------------------------------------------------------------
F: head-to-head, node not in Z -> Blocking

Path is BLOCKED
============================================================

============================================================
Path: E - C - G - I
------------------------------------------------------------
C: tail-to-tail, node not in Z -> Not blocking
G: head-to-tail, node not in Z -> Not blocking

Path is NOT BLOCKED
============================================================

============================================================
Path: E - B - C - G - I
------------------------------------------------------------
B: head-to-head, node not in Z -> Blocking
C: tail-to-tail, node not in Z -> Not blocking
G: head-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: E - F - G - I
------------------------------------------------------------
F: head-to-head, node not in Z -> Blocking
G: tail-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

Conclusion: {'B', 'E'}⍰{'G', 'I'}|{'A'} is False
```

Example 2(c)

Let $X = \{B, E\}, Y = \{G, I\}, Z = \{C, F\}$.

```
X = {'B','E'}
Y = {'G','I'}
Z = {'C','F'}

dag.plot(X, Y, Z,pos=pos, title="d-separation Example 2(c)",
                          figsize=(5,5))
```



Use networkx built-in d-separation analyser.

```
print("\nUsing networkx built-in d-separation analyser")
print(f"  {X}⊥{Y}|{Z} is {dag.is_d_separated(X, Y, Z)}")
```

```
Using networkx built-in d-separation analyser

    {'B','E'} ⊥ {'G','I'}|{'F','C'} is False
```

Do path-by-path analysis.

```
# Path-by-path blocking analysis
dsep, results = dag.path_blocking_analysis(X, Y, Z)
```

```
Path-by-Path Analysis:
============================================================
Path: B - C - E - F - G
------------------------------------------------------------
C: tail-to-tail, node in Z -> Blocking
E: head-to-tail, node not in Z -> Not blocking
F: head-to-head, node in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: B - C - G
------------------------------------------------------------
C: tail-to-tail, node in Z -> Blocking

Path is BLOCKED
============================================================

============================================================
Path: B - E - C - G
------------------------------------------------------------
E: head-to-tail, node not in Z -> Not blocking
C: tail-to-tail, node in Z -> Blocking

Path is BLOCKED
============================================================

============================================================
Path: B - E - F - G
------------------------------------------------------------
E: tail-to-tail, node not in Z -> Not blocking
F: head-to-head, node in Z -> Not blocking

Path is NOT BLOCKED
============================================================

============================================================
Path: B - C - E - F - G - I
------------------------------------------------------------
C: tail-to-tail, node in Z -> Blocking
E: head-to-tail, node not in Z -> Not blocking
F: head-to-head, node in Z -> Not blocking
```

```
G: tail-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: B - C - G - I
------------------------------------------------------------
C: tail-to-tail, node in Z -> Blocking
G: head-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: B - E - C - G - I
------------------------------------------------------------
E: head-to-tail, node not in Z -> Not blocking
C: tail-to-tail, node in Z -> Blocking
G: head-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: B - E - F - G - I
------------------------------------------------------------
E: tail-to-tail, node not in Z -> Not blocking
F: head-to-head, node in Z -> Not blocking
G: tail-to-tail, node not in Z -> Not blocking

Path is NOT BLOCKED
============================================================

============================================================
Path: E - C - G
------------------------------------------------------------
C: tail-to-tail, node in Z -> Blocking

Path is BLOCKED
============================================================

============================================================
Path: E - B - C - G
------------------------------------------------------------
B: head-to-head, node not in Z -> Blocking
C: tail-to-tail, node in Z -> Blocking

Path is BLOCKED
============================================================
```

```
============================================================
Path: E - F - G
------------------------------------------------------------
F: head-to-head, node in Z -> Not blocking

Path is NOT BLOCKED
============================================================

============================================================
Path: E - C - G - I
------------------------------------------------------------
C: tail-to-tail, node in Z -> Blocking
G: head-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: E - B - C - G - I
------------------------------------------------------------
B: head-to-head, node not in Z -> Blocking
C: tail-to-tail, node in Z -> Blocking
G: head-to-tail, node not in Z -> Not blocking

Path is BLOCKED
============================================================

============================================================
Path: E - F - G - I
------------------------------------------------------------
F: head-to-head, node in Z -> Not blocking
G: tail-to-tail, node not in Z -> Not blocking

Path is NOT BLOCKED
============================================================

Conclusion: {'B', 'E'}{'G', 'I'}|{'F', 'C'} is False
```

4. General Risk Preference

4.1 Personal Indifferent Selling Price

We can compute personal indifferent selling price (PISP) for a decision maker under general risk preference with known wealth utility function using the DiscreteDeal class.

```
# Buying and Selling prices under general risk preference
from  DApy.risk import DiscreteDeal
import numpy as np

deal = DiscreteDeal(
    x = np.array([ 10,  -5]),
    p = np.array([0.7, 0.3]))
# The wealth utility function
wealth_uf = lambda w: np.sqrt(w) if w>=0 else -np.sqrt(-w)

# Personal Indifference Selling Prices at different intitial wealths
for w0 in [10, 100, 1000]:
    s = deal.PISP(w0, wealth_uf)
    print(f"w0 = {w0}: PISP = {s:.5f}\n")
```

```
      converged: True
           flag: converged
 function_calls: 4
     iterations: 3
           root: 4.449999999999999
         method: brentq
w0 = 10: PISP = 4.45000

      converged: True
           flag: converged
 function_calls: 4
     iterations: 3
           root: 5.384601430547792
         method: brentq
w0 = 100: PISP = 5.38460

      converged: True
           flag: converged
 function_calls: 4
     iterations: 3
           root: 5.488216792727766
         method: brentq
w0 = 1000: PISP = 5.48822
```

4.2 Personal Indifferent Buying Price

We can compute personal indifferent buying price (PIBP) for a decision maker under general risk preference with known wealth utility function using the DiscreteDeal class.

```
# Personal Indifference Buying Prices at different initial wealth
for w0 in [10, 100, 1000]:
    b = deal.PIBP(w0, wealth_uf)
    print(f"w0 = {w0}: PIBP = {b:.5f}\n")
```

```
      converged: True
           flag: converged
 function_calls: 9
     iterations: 8
           root: 3.7269577158831044
         method: brentq
w0 = 10: PIBP = 3.72696

      converged: True
           flag: converged
 function_calls: 7
     iterations: 6
           root: 5.378192514357238
         method: brentq
w0 = 100: PIBP = 5.37819

      converged: True
           flag: converged
 function_calls: 6
     iterations: 5
           root: 5.488151929048737
         method: brentq
w0 = 1000: PIBP = 5.48815
```

5. Constant Absolute Risk Aversion

Under constant absolute risk aversion (CARA), the decision maker's utility function is either linear (i.e., risk neutral) or exponential in form with a constant risk tolerance or a absolute risk aversion coefficient parameter.

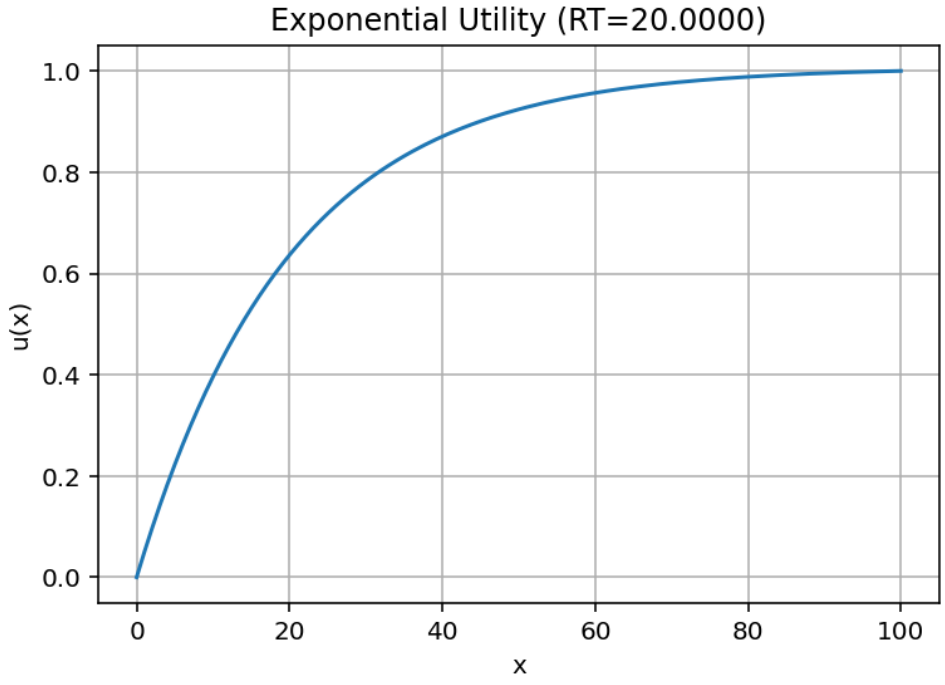
The `ExponUtilityFunction` class provides supports for plotting of the functions and determination of risk tolerance.

```
from  DApy.cara import ExponUtilityFunction
```

5.1 Plotting Exponential Utility Function

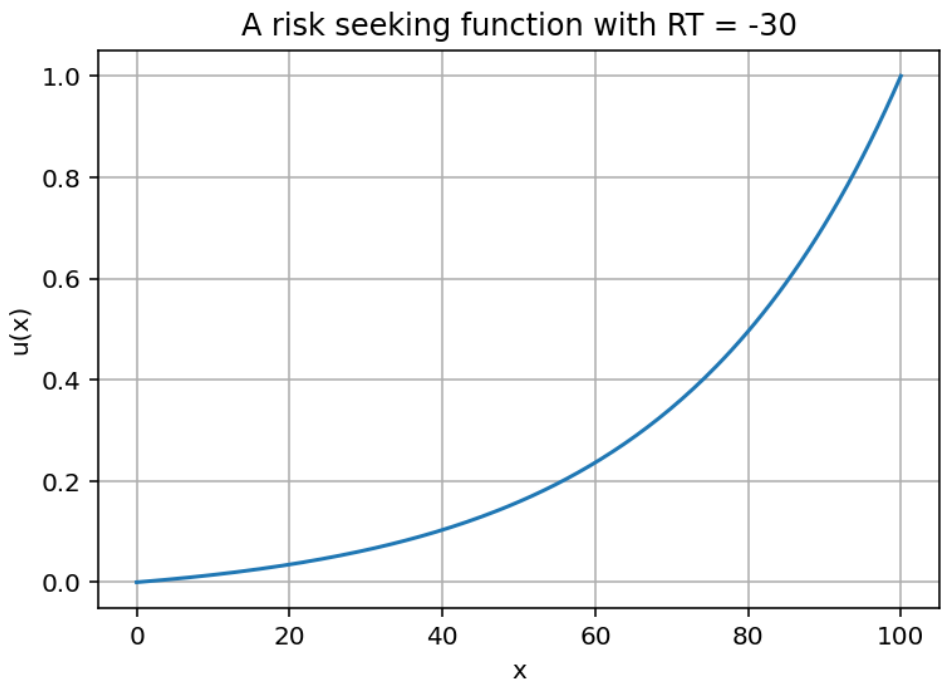
You can plot a single exponential utility function given the risk tolerance and boundary conditions $u(L) = 0$, and $u(H) = 1$.

```
# Create an exponential utility function with risk tolerance = 20.
U1 = ExponUtilityFunction(20)

# Plot it with boundary conditions: u(0)=0, u(100)=1.
U1.plot(L=0, H=100)
```



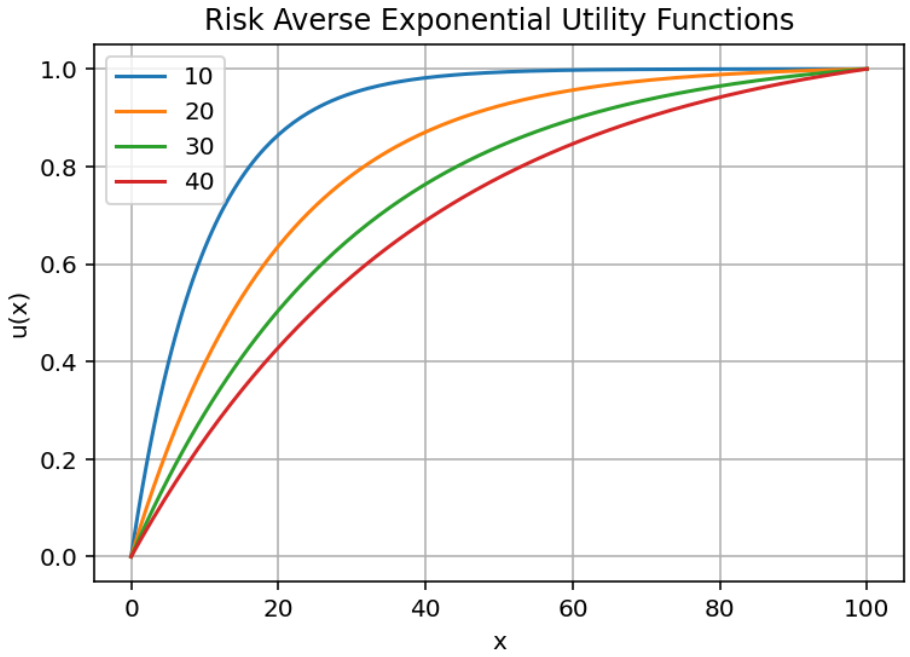
You can plot a risk seeking utility function using a negative risk tolerance and given boundary conditions. You can also change the default title and other Axes properties.

```
# Creat an expontial utility function with risk tolerance = -30.
U2 = ExponUtilityFunction(-30)

# Plot it with boundary conditions and change the default tile.
ax=U2.plot(L=0, H=100)
ax.set_title(f"A risk seeking function with RT = {U2.RT}")
```

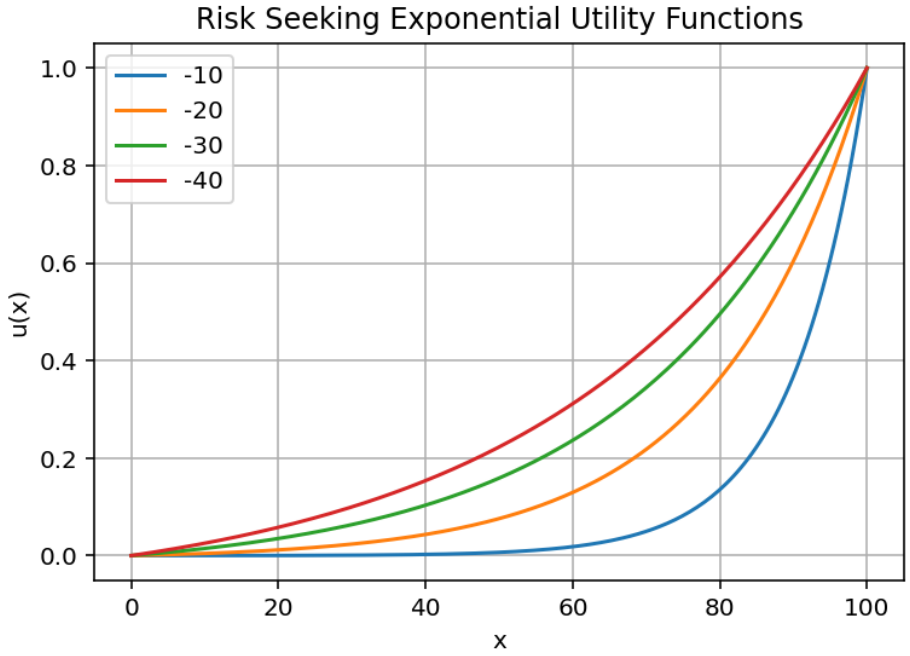


The `DiscreteDeal.plot()` method returns an `Axes` object which you can modify to customise the plot. The following code shows how to plot a family utility functions with a list of risk tolerance.

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
U3 = ExponUtilityFunction()
for r in [10, 20, 30, 40]:
    U3.RT = r
    ax = U3.plot(L=0, H=100, ax=ax)
ax.set_title("Risk Averse Exponential Utility Functions")
ax.legend()
plt.show()
```

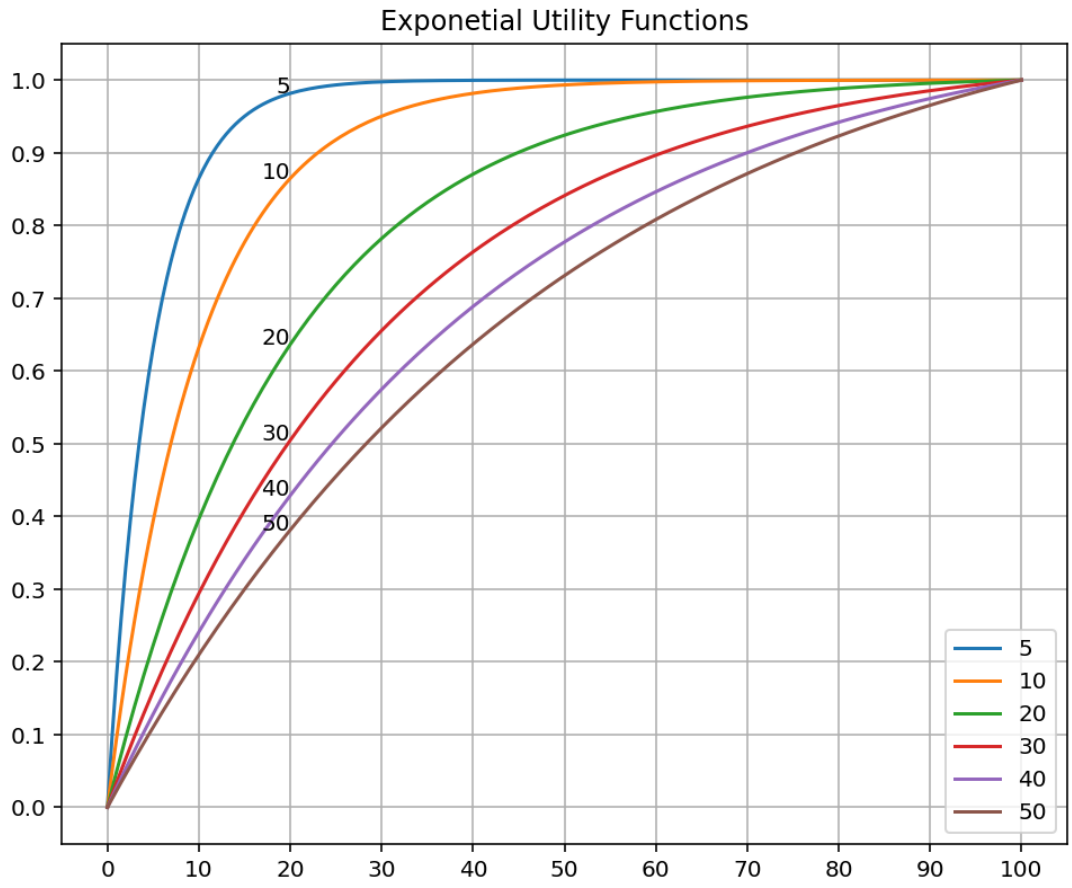



```
fig, ax = plt.subplots()
U4 = ExponUtilityFunction()
for r in [-10, -20, -30, -40]:
    U4.RT = r
    ax = U4.plot(L=0, H=100, ax=ax)
ax.set_title("Risk Seeking Exponential Utility Functions")
ax.legend()
plt.show()
```



For more flexibilty, you can use the `DiscreteDeal.get_function()` method to create a callable utility function with specific boundary condtions. You can use this callable function in any way you like.

```
# Use the callable utility function for plotting.
import numpy as np
L, H = 0, 100
U5 = ExponUtilityFunction()
fig, ax = plt.subplots(figsize=(8,6.4))
x = np.linspace(L, H, 201)
for r in [5, 10, 20, 30, 40, 50]:
    U5.RT = r
    fun = U5.get_function(L=L, H=H)
    ax.plot(x, fun(x), label=r)
    ax.text(20, fun(20), r, ha='right')
ax.set_title("Exponetial Utility Functions")
ax.set_xticks(np.linspace(0,100,11))
ax.set_yticks(np.linspace(0,1,11))
ax.grid()
ax.legend()
plt.show()
```



5.2 Find Risk Tolerance using 50-50 CE Method

The `ExponUtilityFunction` class supports finding risk tolerance using the 50-50 CE method.

Risk Averse Case

When $CE_{0.5} < EV$, the decision maker is risk averse and the risk tolerance is positive.

```
# Risk averse case
L, H, CE50 = 0, 100, 34

# CE50 < (H+L)/2 -> Risk averse -> RT > 0
U1 = ExponUtilityFunction()
```

The equation solver needs either a bracket $[a, b]$ or a single value x_0 for the initial guess.

```
# Provide a bracket
rho = U1.RT_from_CE50(L, H, CE50, bracket=[1, 100])
print(f"risk tolerance = {rho:.4f}")
```

```
risk tolerance = 72.6377
```

```
# Provide a single initial guess
rho = U1.RT_from_CE50(L, H, CE50, x0=50)
print(f"risk tolerance = {rho:.4f}")
```

```
risk tolerance = 72.6377
```

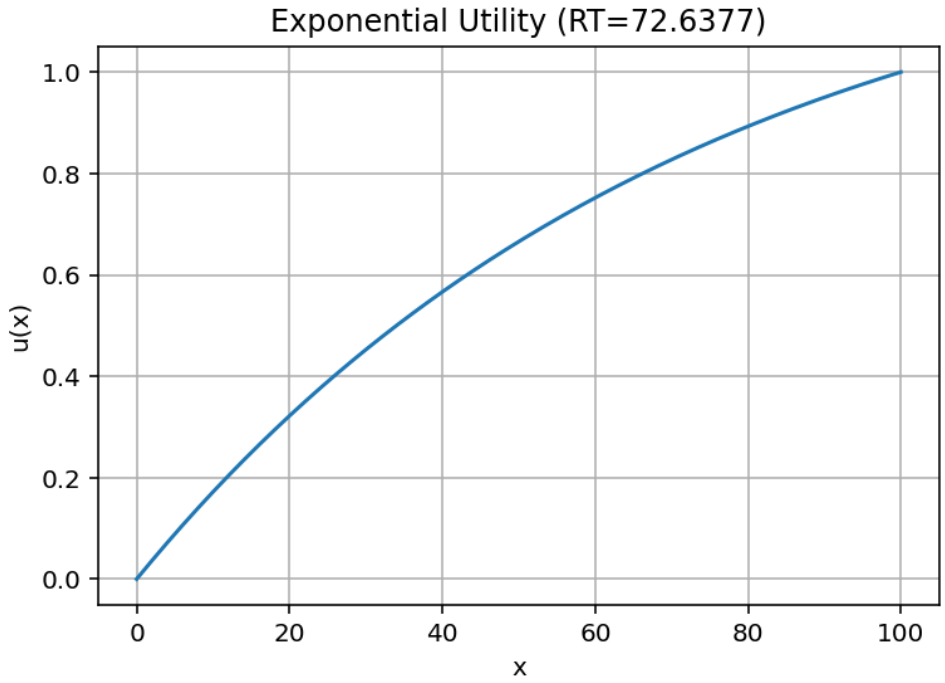
We verify the risk tolerance by computing the CE at utility=0.5.

```
# Check the CE with u(x)=0.5
CE = U1.CE(L, H, 0.5)
print(f"CE(u=0.5)= {CE}")
```

```
CE(u=0.5)= 100.0
```

Plot the utility function.

```
U1.plot(L, H)
```



Risk Seeking Case

```
# Risk seeking case
L, H, CE50 = 0, 100, 65

# CE50 > (H+L)/2 -> Risk seeking -> RT < 0
U2 = ExponUtilityFunction()
```

Providing a bracket as initial guess.

```
# Provide a bracket
rho = U2.RT_from_CE50(L, H, CE50, bracket=[-100, -1])
print(f"risk tolerance = {rho:.4f}")
```

```
risk tolerance = -78.2074
```

Providing a single value as initial guess.

```
# Provide a single initial guess
rho = U2.RT_from_CE50(L, H, CE50, x0=-50)
print(f"risk tolerance = {rho:.4f}")
```

```
risk tolerance = -78.2074
```

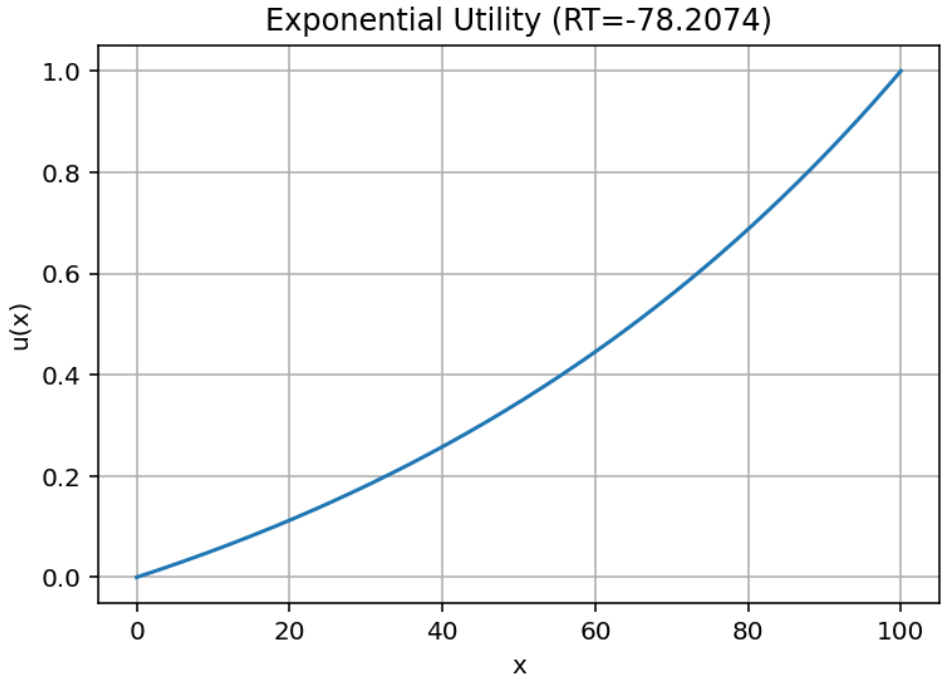
Verify the CE at utility = 0.5

```
# Check the CE with u(x)=0.5
CE = U2.CE(L, H, 0.5)
print(f"CE(u=0.5)= {CE}")
```

```
CE(u=0.5)= 100.0
```

Plot the utility function.

```
U2.plot(L, H)
```



5.3 Buying Price and Selling Price under CARA

We demostrate with a numerical example that the buying and selling price of Kim who has the exponential utility function, for the coin-tossing (0-100) game, are always equal, and independent of the initial wealth.

```
from  DApy.risk import DiscreteDeal
import numpy as np

def wuf(w):
    return 1 - 2**(-w/50)

Coin100 = DiscreteDeal(
    name = 'Coin Toss 0-100',
    x = np.array([0, 100]),
    p = np.array([0.5, 0.5])
    )

results = {}
for w0 in [0, 10, 100, 1000]:
    results[w0] = [Coin100.PISP(w0, wuf), Coin100.PIBP(w0, wuf)]

for k, v in results.items():
    print(f"w0 = {k:4}: PISP = {v[0]:.6f}, PSBP = {v[1]:.6f}")
end{lstlisting}
```

```
converged: True
          flag: converged
function_calls: 9
    iterations: 8
          root: 33.903595255631885
        method: brentq
     converged: True
          flag: converged
function_calls: 9
    iterations: 8
          root: 33.90359525563115
        method: brentq
     converged: True
          flag: converged
function_calls: 9
    iterations: 8
          root: 33.903595255631885
        method: brentq
     converged: True
          flag: converged
function_calls: 9
    iterations: 8
```

```
           root: 33.903595255631146
         method: brentq
      converged: True
           flag: converged
 function_calls: 10
     iterations: 9
           root: 33.903595255631856
         method: brentq
      converged: True
           flag: converged
 function_calls: 9
     iterations: 8
           root: 33.903595255631174
         method: brentq
      converged: True
           flag: converged
 function_calls: 8
     iterations: 7
           root: 33.90359525896935
         method: brentq
      converged: True
           flag: converged
 function_calls: 8
     iterations: 7
           root: 33.90359525452961
         method: brentq

w0 =    0: PISP = 33.903595, PSBP = 33.903595
w0 =   10: PISP = 33.903595, PSBP = 33.903595
w0 =  100: PISP = 33.903595, PSBP = 33.903595
w0 = 1000: PISP = 33.903595, PSBP = 33.903595
```

6. Fitting Probability Distributions to Data

6.1 Fitting Continuous Distributions to Data

The `scipy.stats` module directly supports the fitting of continuous distributions to data using the maximum likelihood method (MLE) or method of moments (MM). The `DApy.probability.FitPDF.fit()` method is based on `scipy.stats`.

```
# Fit continuous distribution to data with scipy.stats
from DApy.probability import FitPDF
from DApy.utils import read_data_from_excel

# Read data from data file.  Replace this with your own data loader
file_name = "7.4.3_example_data.xlsx"
sheet_name = "data_1"
data = read_data_from_excel(file_name, sheet_name)
```

```
Data size = 100
```

You should check that the data was loaded correctly before proceeding with the fit.

```
# Create the fitter
model = FitPDF(data)
# Fit the data using MLE (default). Can also be MM.
model.fit(method='MLE')
```

```
fitting norm
fitting lognorm
fitting expon
fitting gamma
fitting weibull_min
fitting weibull_max
fitting beta
fitting logistic
fitting gumbel_r
fitting gumbel_l
fitting fisk
fitting laplace
fitting rayleigh
fitting cauchy
```

```
fitting pareto
fitting dweibull
fitting genextreme
fitting genlogistic
fitting gennorm
fitting halfnorm
fitting invgauss
fitting powerlaw
fitting triang
fitting uniform
fitting wald
fitting burr

Fitted 26 distributions successfully using MLE
```

Sort and show the top 20 results base on KS criterion.

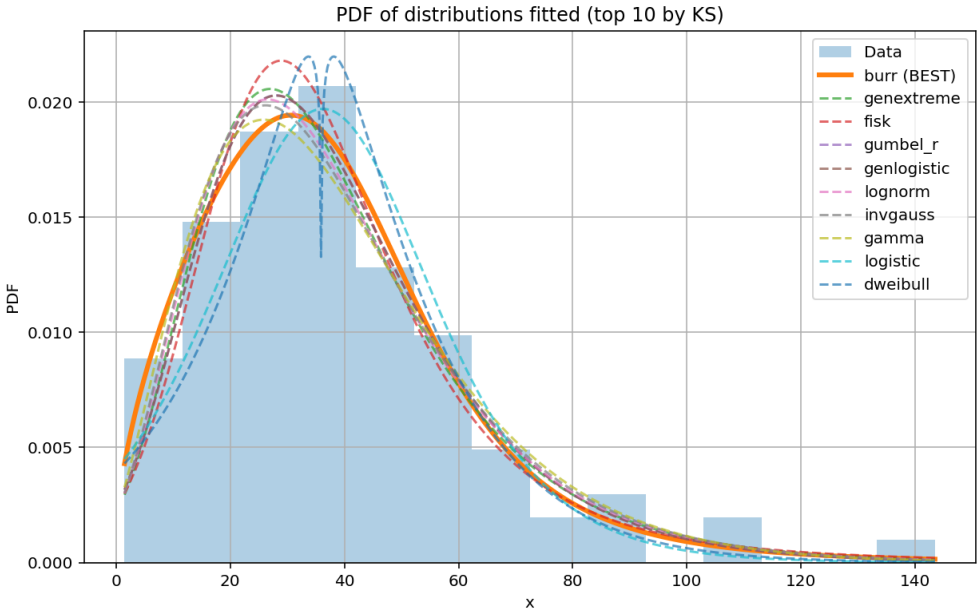
```
# sort = 'KS','AIC','BIC' or 'RMSE_pdf'
model.results(sort='KS', top_k=20)
```

```
   distribution                                          params  k_params  \
0          burr  (4.22354262066905, 0.3750679177723734, -0.3401...         4
1     genextreme  (-0.03794583325239832, 27.658238821980085, 17....         3
2          fisk  (4.323221940268919, -17.96671595019074, 52.317...         3
3      gumbel_r            (28.03306142496813, 18.127671213467035)         2
4   genlogistic  (383.39819602181194, -79.68142660990009, 18.11...         3
5       lognorm  (0.39101141371588294, -20.436305267152363, 54....         3
6      invgauss  (0.1545954054872528, -22.07107210705672, 392.8...         3
7         gamma  (3.44717666430875, -5.504349155214378, 12.8138...         3
8      logistic            (36.21683580667891, 12.685398572525827)         2
9      dweibull  (1.1068536961905613, 35.91601423207297, 18.238...         3
10  weibull_min  (1.3564567090283006, 8.818653216921348, 33.544...         3
11     gennorm  (1.0832279490468268, 34.5673101161983, 19.4247...         3
12        beta  (3.6138205026542556, 18.18404568590035, -9.462...         4
13     laplace                       (34.370000000000005, 17.4611)         2
14     rayleigh            (-3.589560950167827, 34.44588157582716)         2
15       cauchy             (33.84728478720986, 11.93458509633832)         2
16         norm            (38.667300000000004, 24.23623769709317)         2
17     halfnorm                             (1.48, 44.38795443586019)         2
18         wald            (-3.259948066822867, 50.32528267794876)         2
19     gumbel_l             (52.13002452185519, 32.72655290685559)         2
20       pareto  (461982154.2806587, -17179869182.519993, 17179...         3
21        expon                             (1.48, 37.18730000000001)         2
22     powerlaw  (0.4985026950588382, 1.4799999999999998, 141.9...         3
23       triang  (0.09661442383745106, -2.3761279638830644, 148...         3
24      uniform                            (1.48, 141.98000000000002)         2
25  weibull_max  (0.2078074447616376, 143.46000000000004, 1.624...         3

        loglik        KS          KS_p          AIC          BIC  RMSE_pdf
```

```
0   -447.176649  0.041298  9.930111e-01   902.353298   912.773979  0.001026
1   -448.253972  0.044278  9.847155e-01   902.507944   910.323454  0.001367
2   -448.402172  0.045694  9.789946e-01   902.804344   910.619854  0.001376
3   -448.406972  0.047548  9.695060e-01   900.813944   906.024285  0.001256
4   -448.414120  0.047560  9.694398e-01   902.828241   910.643751  0.001253
5   -448.252883  0.048185  9.656965e-01   902.505766   910.321276  0.001367
6   -448.356360  0.051456  9.414478e-01   902.712719   910.528230  0.001393
7   -448.432826  0.056894  8.840725e-01   902.865653   910.681163  0.001417
8   -455.654285  0.060749  8.321956e-01   915.308569   920.518910  0.001751
9   -454.844187  0.066277  7.465584e-01   915.688373   923.503884  0.001818
10         -inf  0.067509  7.263050e-01          inf          inf  0.003204
11  -455.231083  0.074460  6.094109e-01   916.462165   924.277676  0.001849
12  -450.035592  0.074621  6.066996e-01   908.071185   918.491866  0.001371
13  -455.312273  0.076020  5.833143e-01   914.624546   919.834886  0.002015
14  -450.710280  0.091560  3.501873e-01   905.420559   910.630900  0.001564
15  -464.382203  0.112445  1.476849e-01   932.764406   937.974746  0.002768
16  -460.678747  0.117047  1.191260e-01   925.357494   930.567835  0.002532
17  -451.875949  0.125795  7.729652e-02   907.751898   912.962238  0.003531
18  -463.251810  0.135484  4.615763e-02   930.503620   935.713960  0.004341
19  -489.955684  0.191636  1.085996e-03   983.911368   989.121708  0.004932
20  -461.596732  0.194551  8.624480e-04   929.193464   937.008975  0.005373
21  -461.596731  0.194551  8.624476e-04   927.193461   932.403801  0.005373
22  -464.582528  0.235924  2.220643e-05   935.165055   942.980566  0.006197
23  -463.934898  0.242794  1.126560e-05   933.869796   941.685306  0.004066
24  -495.568620  0.456697  1.380667e-19   995.137241  1000.347581  0.007001
25  -740.771413  0.841686  3.809303e-80  1487.542827  1495.358337  0.010371
```

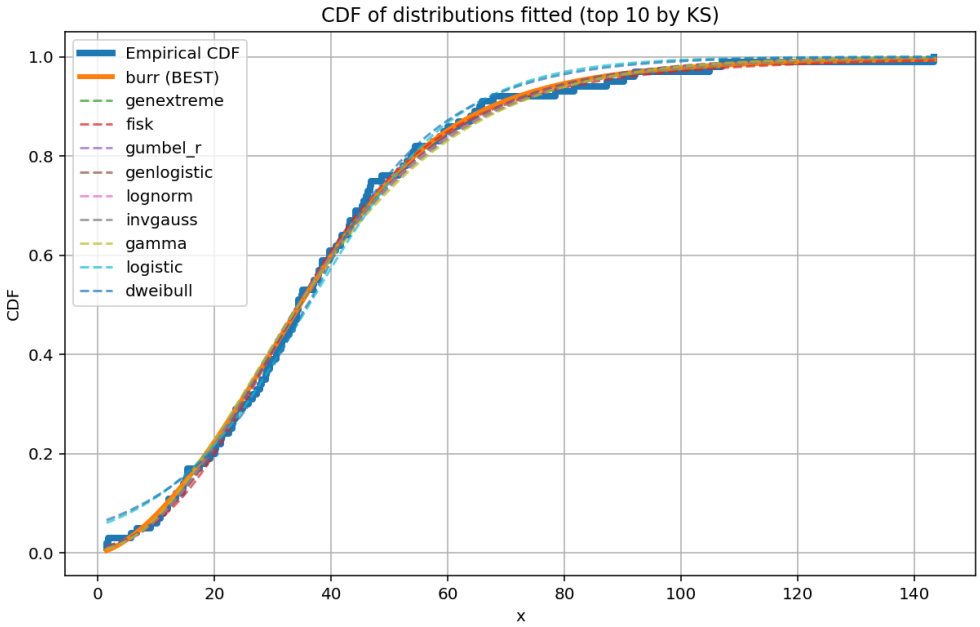
Plot the PDF of the top 10 results.

```
model.plot_pdf(sort='KS',top_k=10)  # top_k = 10 (default)
```



Plot the CDF of the top 10 results.

```
model.plot_cdf(sort='KS',top_k=10)
```



Extract the results for the lognormal distribution (index = 5).

```
params = results_KS.iloc[5, results_KS.columns.get_loc('params')]
for p in params:
    print(float(p))
```

```
0.39101141371588294
-20.436305267152363
54.74077290040369
```

Save all the results to an Excel file.

```
model.save_results_to_excel("outputs1.xlsx", sort='KS')
```

6.2 Fitting Discrete Distributions to Data

`scipy.stats` does not provide direct support for fitting discrete distributions to data. We will use numerical optimization of the MLE to do the fitting. The method depends very much on providing good initial guesses for the optimization solver.

```
# Fit PMF to data example
from DApy.probability import FitPMF
from DApy.utils import read_data_from_excel

# Read data from data file.  Replace this with your own data loader
file_name = "7.4.3_example_data.xlsx"
sheet_name = "data_2"
data = read_data_from_excel(file_name, sheet_name)
```

```
Data size = 100
```

You should check that the data was loaded correctly before proceeding with the fit.

```
# Create the fitter
model = FitPMF(data)
# Do the fit
model.fit()
```

```
Fitting geom.
Fitting nbinom.
Fitting binom.
Fitting poisson.
Fitting logser.
Fitting zipf.
Fitting dlaplace.
Fitting planck.
Fitting skellam.
Fitting bernoulli.
     Did not converge.
Fitting hypergeom.
     Did not converge.
Fitting randint.
     Did not converge.
Fitting boltzmann.
     Did not converge.

Fitted 9 distributions successfully using MLE with solver L-BFGS-B
```

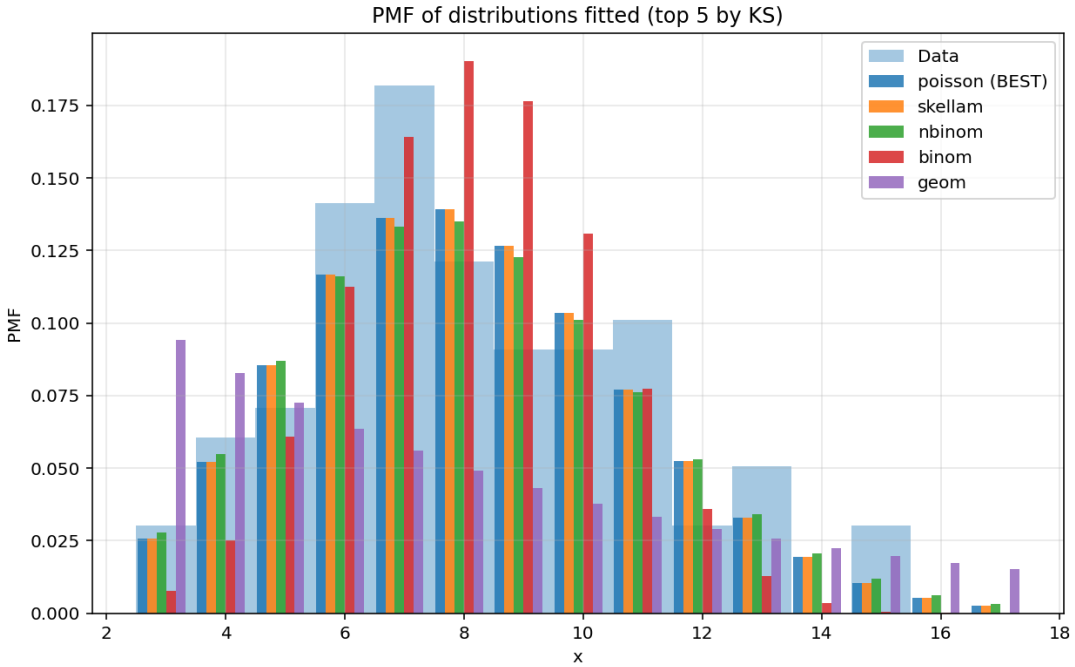
Sort by 'KS' and show the top 5 distributions.

```
# sort can be 'KS','AIC' or 'BIC'
results_KS=model.results(sort='KS', top_k=5)
```

```
Top 5 fitted distribution by KS
  distribution                                      params      loglik  \
0      poisson                                    (8.18,) -245.694110
1      skellam                (8.179995160168845, 1e-06) -245.694111
2       nbinom  (128.37162384420148, 0.9400959181335145) -245.708667
3        binom               (17.0, 0.48117647058823526) -265.139551
4         geom                    (0.12224938865770929,) -303.791222

          AIC         BIC        KS          KS_p  converged
0  493.388220  495.993390  0.121887  5.614611e-02       True
1  495.388222  500.598563  0.121888  5.614399e-02       True
2  495.417333  500.627674  0.129724  3.639270e-02       True
3  534.279102  539.489442  0.170015  2.627227e-03       True
4  609.582444  612.187615  0.378978  3.900936e-14       True

Best distribution: poisson
Parameters: 8.18
```

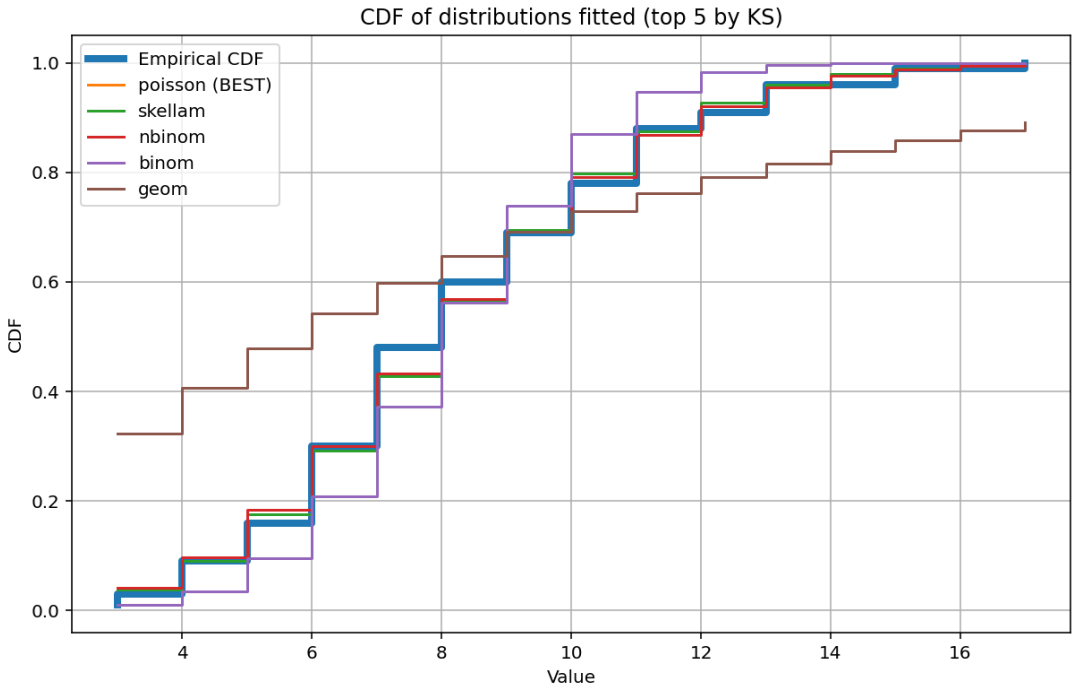
Compare the PMF of the top 5 distributions.

```
model.plot_pmf(sort='KS',top_k=5)  # top_k=10 (default)
```



Compare the PMF of the top 5 distributions

```
model.plot_cdf(sort='KS',top_k=5)
```



Extract the results for binomial distribution (index = 3)

```
params = results_KS.iloc[3, results_KS.columns.get_loc('params')]
for p in params:
    print(float(p))
```

```
17.0
0.48117647058823526
```

Save all the results to an Excel file.

```
model.save_results_to_excel("outputs2.xlsx", sort='KS')
```

6.3 Fitting Normal Distribution to Percentile Data

The `DApy.probability.Fit_norm_percentiles` method can be used to fit a normal distribution to data in percentile format.

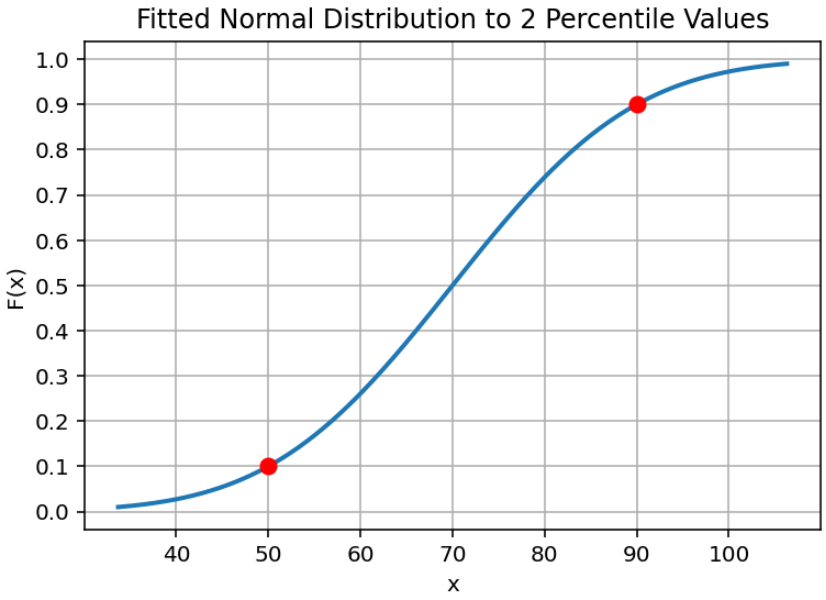
6.3.1 Fitting 2 Percentile Values

When the data comprises only two percentiles values, we can compute the two parameters μ and σ of the normal distribution in closed-form.

```
# Fit a normal distribution to 2 percentile values (2026 06 13)
from DApy.probability import Fit_norm_percentiles
import matplotlib.pyplot as plt

# Two points (exact closed-form solution)
xs = [50,   90]
ps = [0.1, 0.9]

# Fit the distribution
norm_fit2p = Fit_norm_percentiles(xs, ps)
mu, sigma = norm_fit2p.fit()
print(f"mu    = {mu:.2f}")
print(f"sigma = {sigma:.2f}")

if not(mu is None or sigma is None):
# Plot the fitted PDF
ax = norm_fit2p.plot()
ax.set_title("Fitted Normal Distribution")
ax.grid()
plt.show()
```

```
mu = 70.00, sigma=15.61
```

```
# Plot the fitted PDF
ax = fit2p.plot()
ax.set_title("Fitted Normal Distribution to 2 Percentiles")
ax.grid()
plt.show()
```



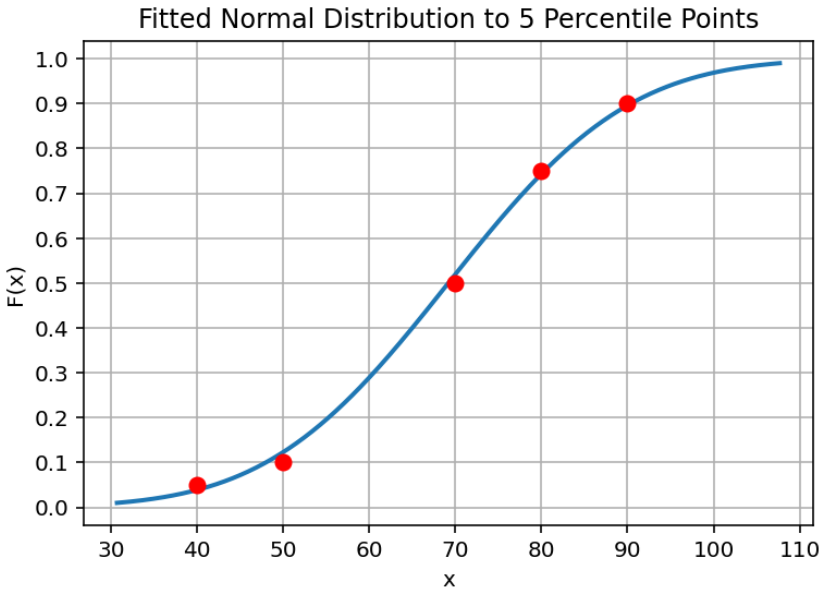
6.3.2 Fitting More Than 2 Percentile Values

When there are more than 2 percentile values to fit, `DApy.probability.Fit_norm_percentiles` uses ordinary least squares method to fit the two parameters μ and σ. A third value $r_squared$ is returned indicating the goodness of the fit.

```
# Fit a normal distribution to more than 2 percentile values
from DApy.probability import Fit_norm_percentiles
import matplotlib.pyplot as plt

xs = [40,   50,  70,  80,   90 ]
ps = [0.05, 0.1, 0.5, 0.75, 0.9]

# Fit the distribution
norm_fit5p = Fit_norm_percentiles(xs, ps)
mu, sigma, r_sq = norm_fit5p.fit()
print(f"mu    = {mu:.4f}")
print(f"sigma = {sigma:.4f}")
print(f"r_sq  = {r_sq:.4f}")

# Plot the fitted PDF
ax = norm_fit5p.plot()
ax.set_title("Fitted Normal Distribution to 5 Percentil Values")
ax.grid()
plt.show()
```

```
mu    = 69.2094
sigma = 16.5371
r_sq  = 0.9948
```

```
# Plot the fitted PDF
ax = fit5p.plot()
ax.set_title("Fitted Normal Distribution to 5 Percentile Points")
ax.grid()
plt.show()
```



7. The Exxoff Problem

7.1 Exoff One-Way Range Sensitivity Analysis

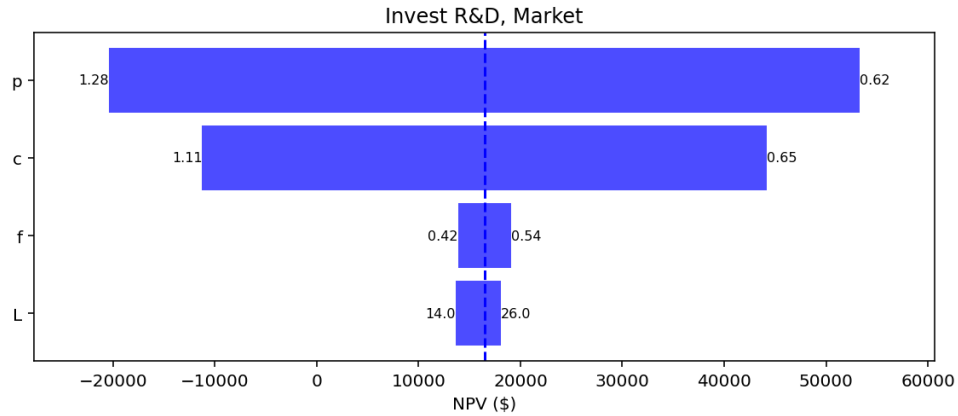
Set up the data and define the objective (NPV) functions for sensitivity analysis.

```
# Exoff Problem One-Way Range Sensitviity Analysis
from DApy.sensit import OneWaySensitivity
import pandas as pd
import numpy_financial as npf

# Set up the data.
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 ] }
marr = 0.1
scenarios = ["Invest R&D, Market",
             "Invest R&D, Don't Market",
             "Don't Invest"]
def npv_1(c, p, f, L):
    return -npf.pv(marr, 5, 0, -7-npf.pv(marr, L, 50*f*(1-p*c)))*1000
def npv_2(c, p, f, L):
    return -npf.pv(marr, 5, 0, -7)*1000
def npv_3(c, p, f, L):
    return 0

obj_func = [npv_1, npv_2, npv_3]
colors = ['blue','orange','green']
value_label = "NPV ($)"
```

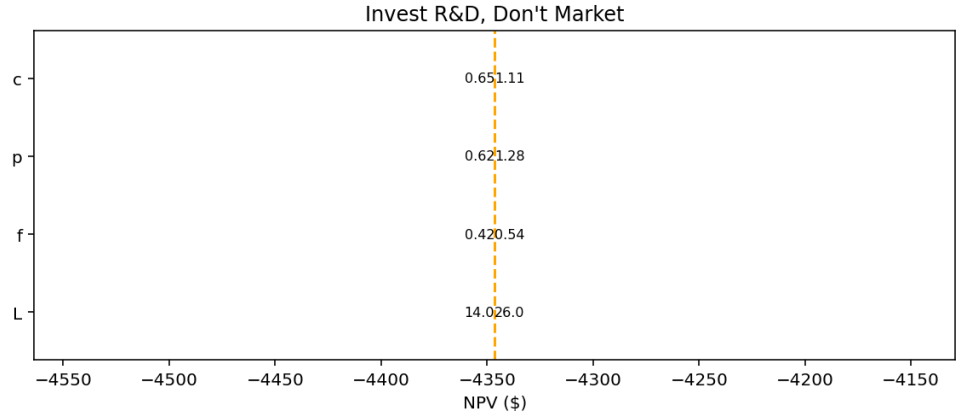
Plot individual tornados.

```
table_dfs = []
pd.set_option('display.max_columns', None)
for i, name in enumerate(scenarios):
    m = OneWaySensitivity(obj_func[i], data, label=name)
    df = m.sensit()
    print(f"\nSenario: {name}")
    print(f"Base objective value = {m.base_result}")
    print(df)
    m.plot_tornado(colors[i], obj_label=value_label)
    table_dfs.append(df)
```

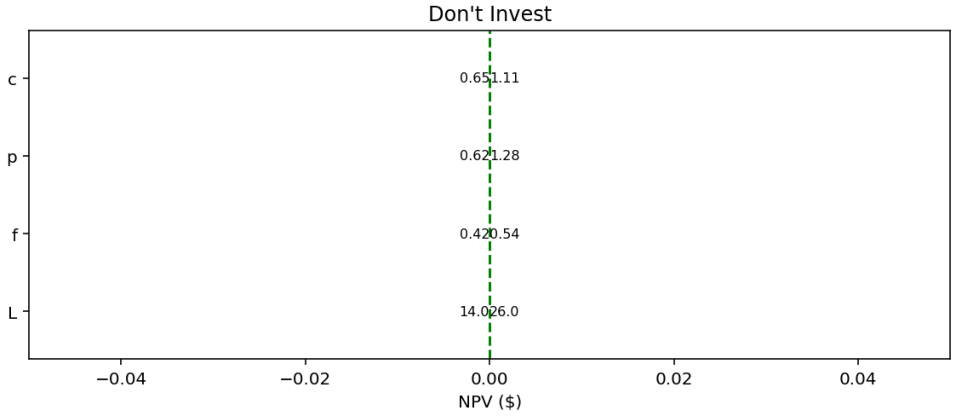
```
Scenario: Invest R&D, Market
Base objective value = 16460.243525945018
             low   base   high         f_low        f_base        f_high  \
variable
c           0.65   0.88   1.11  44181.355563  16460.243526 -11260.868511
p           0.62   0.95   1.28  53303.314169  16460.243526 -20382.827117
f           0.42   0.48   0.54  13859.406928  16460.243526  19061.080124
L          14.00  20.00  26.00  13657.339495  16460.243526  18042.409780

                 swing
variable
c          55442.224074
p          73686.141286
f           5201.673197
L           4385.070284
```




```
Scenario: Invest R&D, Don't Market
Base objective value = -4346.4492614140845
             low   base   high        f_low       f_base       f_high  swing
variable
c           0.65   0.88   1.11 -4346.449261 -4346.449261 -4346.449261    0.0
p           0.62   0.95   1.28 -4346.449261 -4346.449261 -4346.449261    0.0
f           0.42   0.48   0.54 -4346.449261 -4346.449261 -4346.449261    0.0
L          14.00  20.00  26.00 -4346.449261 -4346.449261 -4346.449261    0.0
```

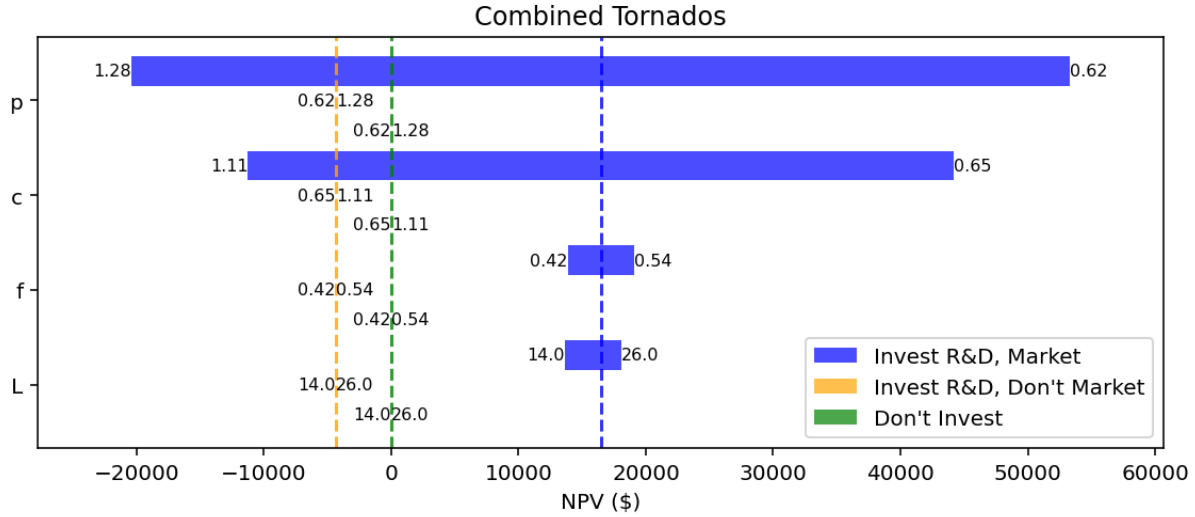


```
Scenario: Don't Invest
Base objective value = 0
             low   base   high  f_low  f_base  f_high  swing
variable
c           0.65   0.88   1.11      0       0       0      0
p           0.62   0.95   1.28      0       0       0      0
f           0.42   0.48   0.54      0       0       0      0
L          14.00  20.00  26.00      0       0       0      0
```



Plot combined tornados

```
OneWaySensitivity.combine_tornadoes(table_dfs, labels=scenarios,
    colors=colors, obj_label=value_label)
```



7.2 Exxoff Stochastic Dominance Analysis

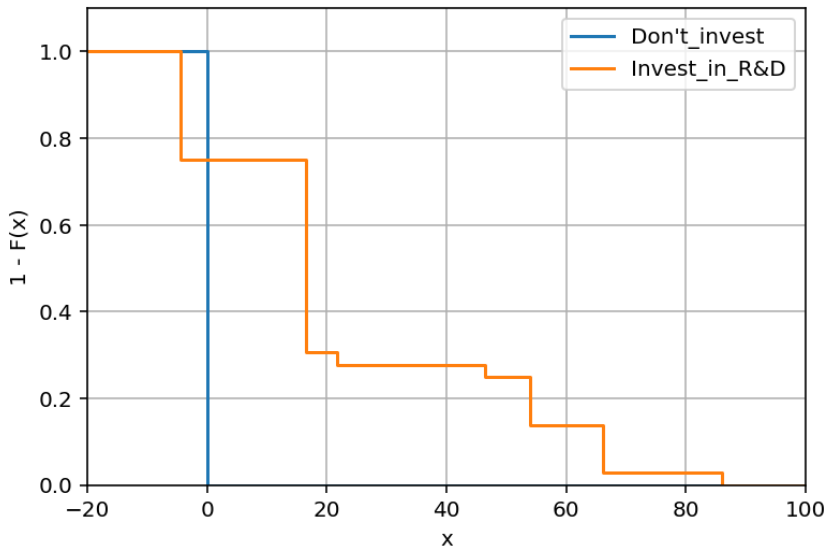
Set up the risk profile data.

```
# Exxoff Problem Stochastic Dominance Analysis
from  DApy.risk import DiscreteDeal

# Set up the data.
Invest = DiscreteDeal (
    name='Invest_in_R&D',
    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])

No_invest = DiscreteDeal(
    name="Don't_invest",
    x=[0],
    p=[1])
```

Determine if No_invest first order dominates Invest.

```
xlim= (-20, 100)
DiscreteDeal.first_order_SD(No_invest, Invest, xlim=xlim,
        plot_epf=True, plot_cdf=False)
```

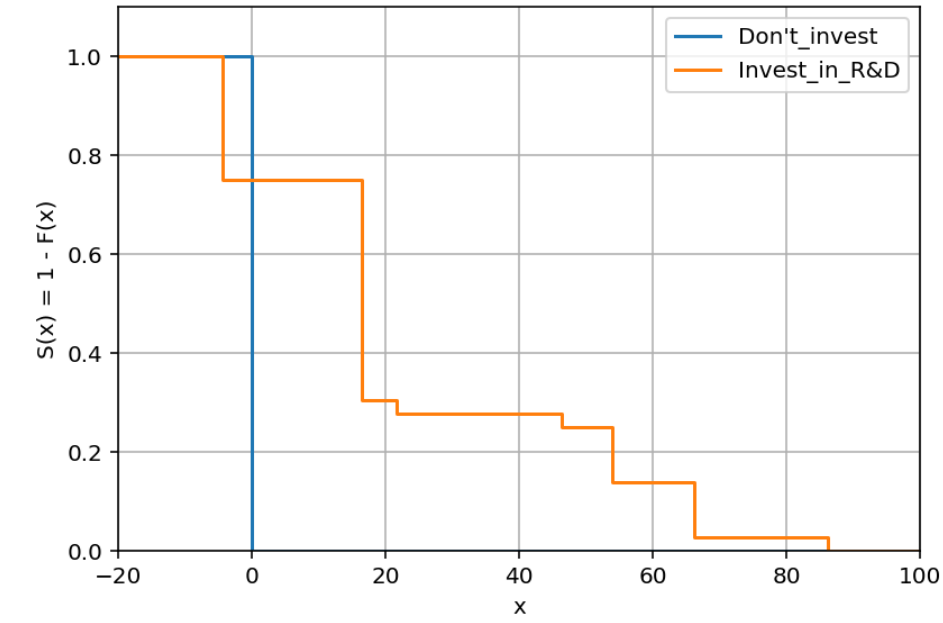
```
Don't_invest does not 1SD Invest_in_R&D
```

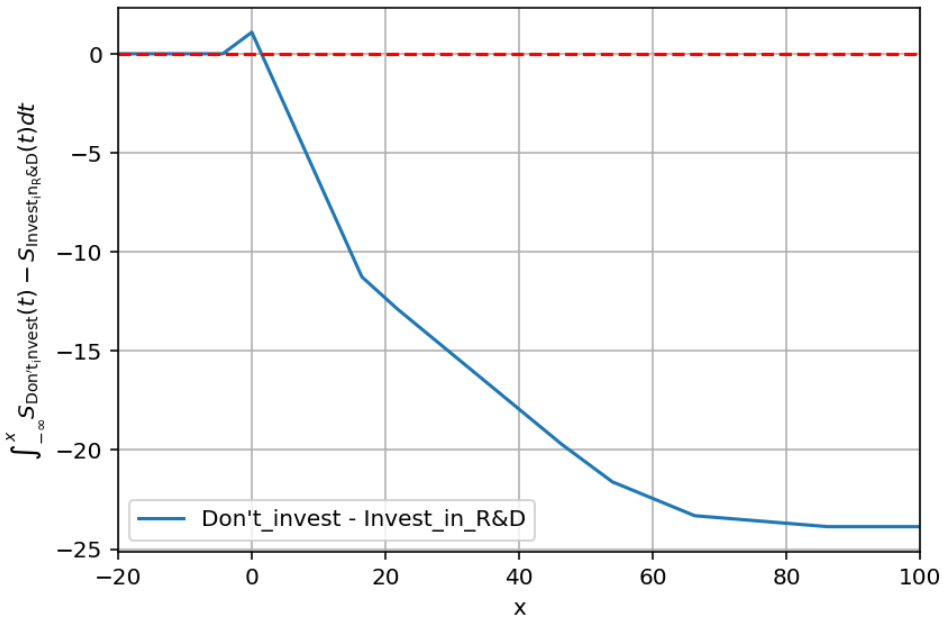


Determine if No_invest second order dominates Invest.

```
DiscreteDeal.second_order_SD(No_invest, Invest, xlim=xlim, plot=True)
```

```
Don't_invest does not 2SD Invest_in_R&D
```





8. The LIM Problem

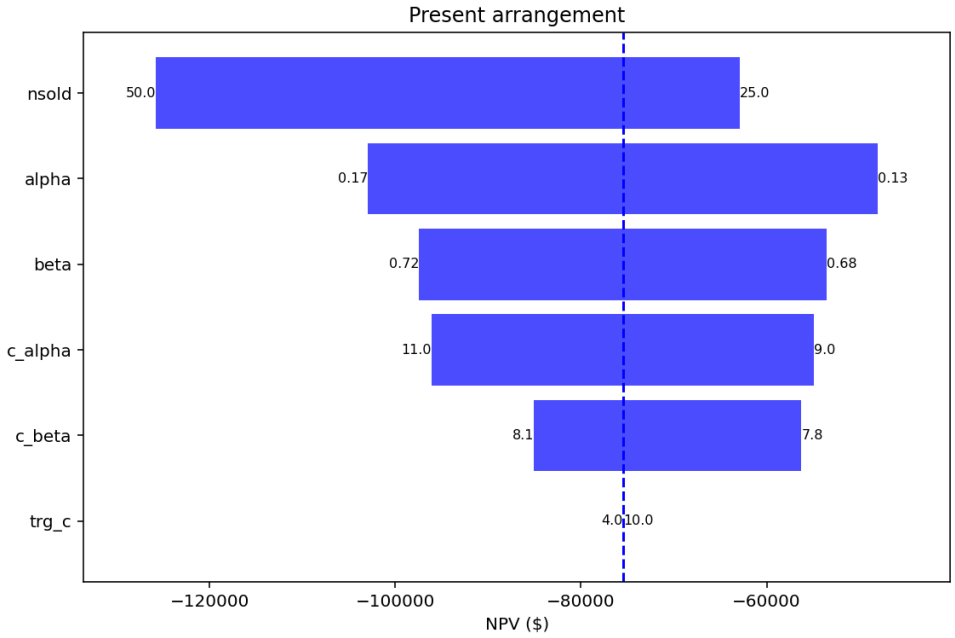
8.1 LIM One-Way Range Sensitivity Analysis

Set up the data and objective (NPV) functions for sensitivity analysis.

```
# LIM Problem One-Way Range Sensitivity Analysis
from DApy.sensit import OneWaySensitivity
import pandas as pd
import numpy_financial as npf

# Set up the data
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]}

marr = 0.03

scenarios = ['Present arrangement',
             'Train user',
             'Contract IPX',
             'Withdraw from market']

def npv_1(nsold, c_alpha, alpha, c_beta, beta, trg_c):
    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):
    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):
    return (nsold*30 - max(750, 23*nsold))*1000
def npv_4(nsold, c_alpha, alpha, c_beta, beta, trg_c):
    return -25.0*1000

obj_func = [npv_1, npv_2, npv_3, npv_4]
colors = ['blue','orange','green','black']
value_label = "NPV ($)"
```

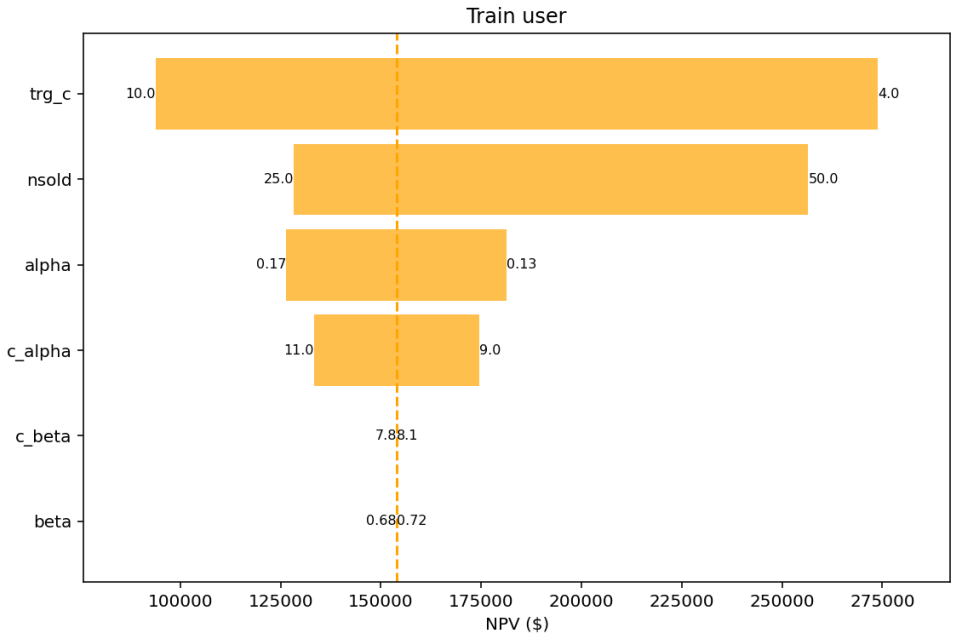
Plot individual tornados.

```
dfs = []
pd.set_option('display.max_columns', None)
for i, name in enumerate(scenarios):
    m = OneWaySensitivity(obj_func[i], data, label=name)
    df = m.sensit()
    m.plot_tornado(colors[i], obj_label=value_label)
    print(f"\nScenario: {name}")
    print(f"Base objective value = {m.base_result}")
    print(df)
    dfs.append(df)
```

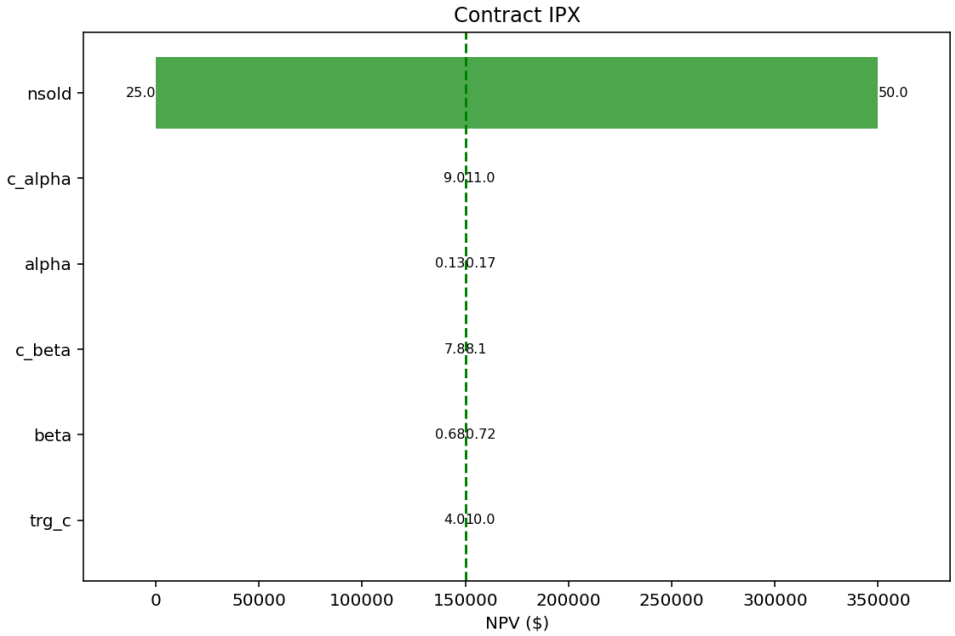



```
Scenario: Present arrangement
Base objective value = -75477.63087243616
             low   base   high         f_low        f_base         f_high  \
variable
nsold      25.00  30.00  50.00 -62898.025727 -75477.630872 -125796.051454
c_alpha     9.00  10.00  11.00 -54868.948530 -75477.630872  -96086.313215
alpha       0.13   0.15   0.17 -47999.387749 -75477.630872 -102955.873996
c_beta      7.80   8.00   8.10 -56242.860686 -75477.630872  -85095.015966
beta        0.68   0.70   0.72 -53495.036374 -75477.630872  -97460.225371
trg_c       4.00   8.00  10.00 -75477.630872 -75477.630872  -75477.630872

                 swing
variable
nsold     62898.025727
c_alpha   41217.364685
alpha     54956.486246
```

```
c_beta    28852.155279
beta      43965.188997
trg_c         0.000000
```

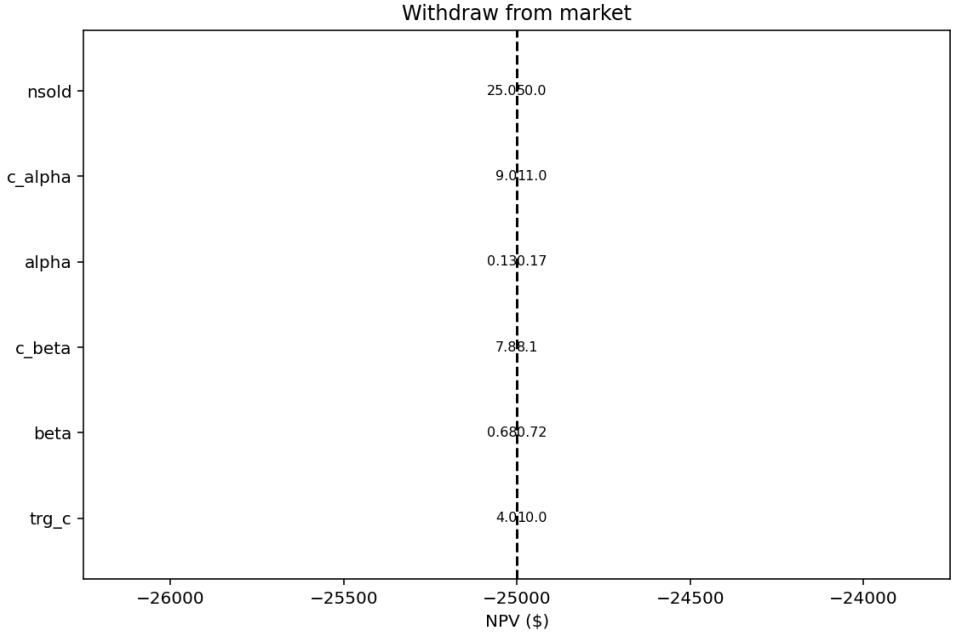



```
Scenario: Train user
Base objective value = 153913.17657624587
            low   base   high          f_low         f_base         f_high  \
variable
nsold     25.00  30.00  50.00  128260.980480  153913.176576  256521.960960
c_alpha    9.00  10.00  11.00  174521.858919  153913.176576  133304.494234
alpha      0.13   0.15   0.17  181391.419699  153913.176576  126434.933453
c_beta     7.80   8.00   8.10  153913.176576  153913.176576  153913.176576
beta       0.68   0.70   0.72  153913.176576  153913.176576  153913.176576
trg_c      4.00   8.00  10.00  273913.176576  153913.176576   93913.176576

                 swing
variable
nsold     128260.980480
c_alpha    41217.364685
alpha      54956.486246
c_beta         0.000000
beta           0.000000
trg_c     180000.000000
```



Scenario: Contract IPX
Base objective value = 150000

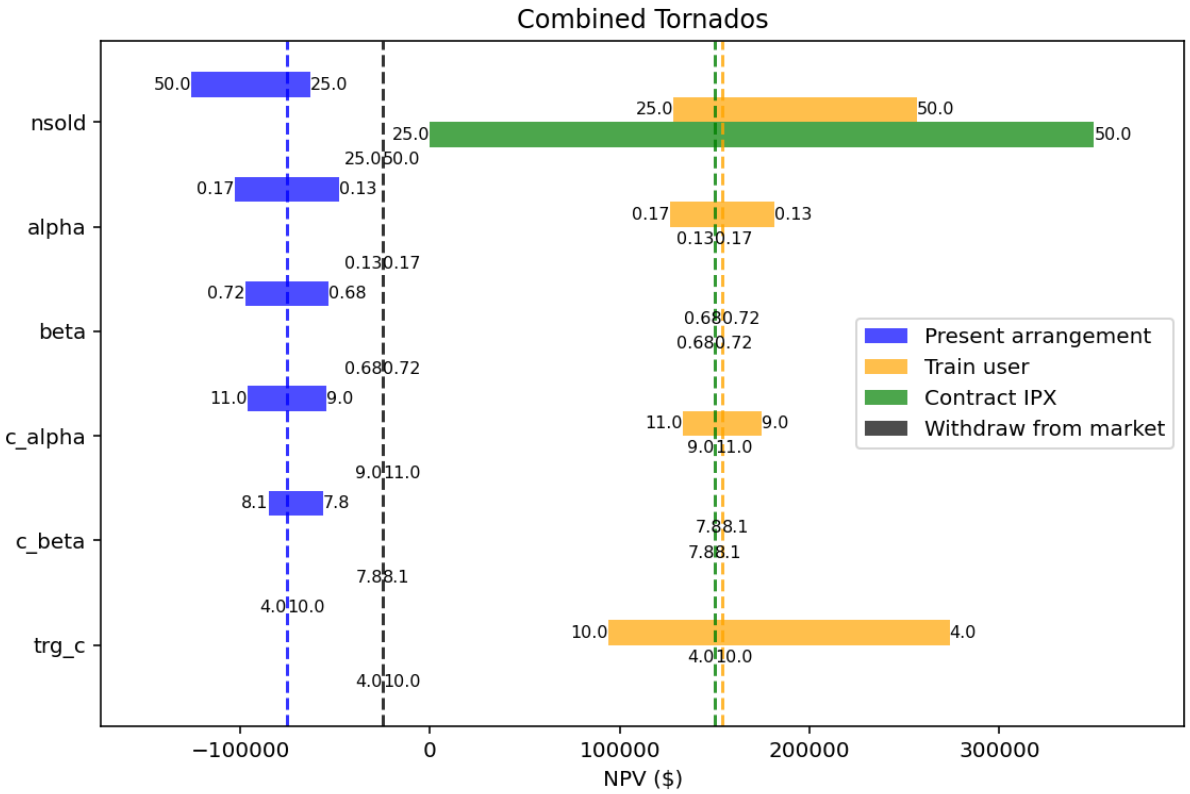
variable	low	base	high	f_low	f_base	f_high	swing
nsold	25.00	30.00	50.00	0	150000	350000	350000
c_alpha	9.00	10.00	11.00	150000	150000	150000	0
alpha	0.13	0.15	0.17	150000	150000	150000	0
c_beta	7.80	8.00	8.10	150000	150000	150000	0
beta	0.68	0.70	0.72	150000	150000	150000	0
trg_c	4.00	8.00	10.00	150000	150000	150000	0



```
Scenario: Withdraw from market
Base objective value = -25000.0
            low   base   high    f_low   f_base   f_high  swing
variable
nsold     25.00  30.00  50.00 -25000.0 -25000.0 -25000.0    0.0
c_alpha    9.00  10.00  11.00 -25000.0 -25000.0 -25000.0    0.0
alpha      0.13   0.15   0.17 -25000.0 -25000.0 -25000.0    0.0
c_beta     7.80   8.00   8.10 -25000.0 -25000.0 -25000.0    0.0
beta       0.68   0.70   0.72 -25000.0 -25000.0 -25000.0    0.0
trg_c      4.00   8.00  10.00 -25000.0 -25000.0 -25000.0    0.0
```

Plot combined tornados

```
# Plot combined tornados
OneWaySensitivity.combine_tornadoes(dfs, labels=scenarios,
    colors=colors, obj_label=value_label)
```



8.2 LIM Stochastic Dominance Analysis

```
# LIM Problem Stochastic Dominance Analysis
from  DApy.risk import DiscreteDeal
```

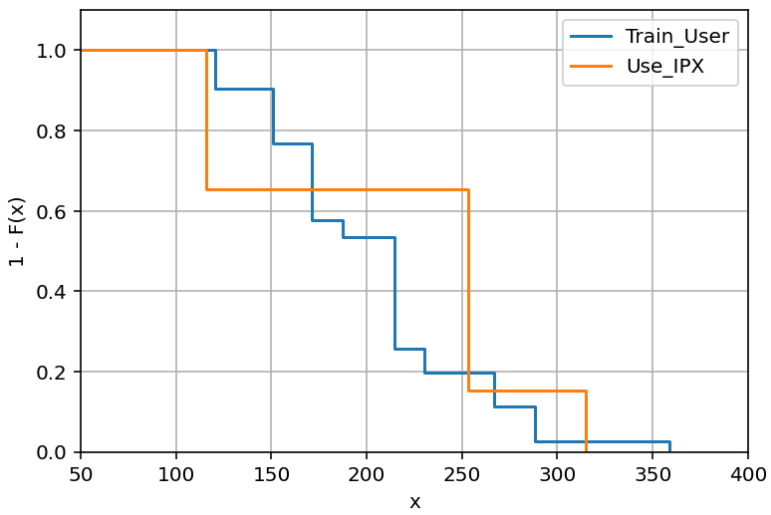
Set up the data.

```
Train = DiscreteDeal(
    name = 'Train_User',
    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 ] )

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

Check if Train User 1st order dominates Use IPX.

```
xlim= (50, 400)
DiscreteDeal.first_order_SD(Train, IPX, xlim=xlim,
                            plot_epf=True, plot_cdf=False)
```

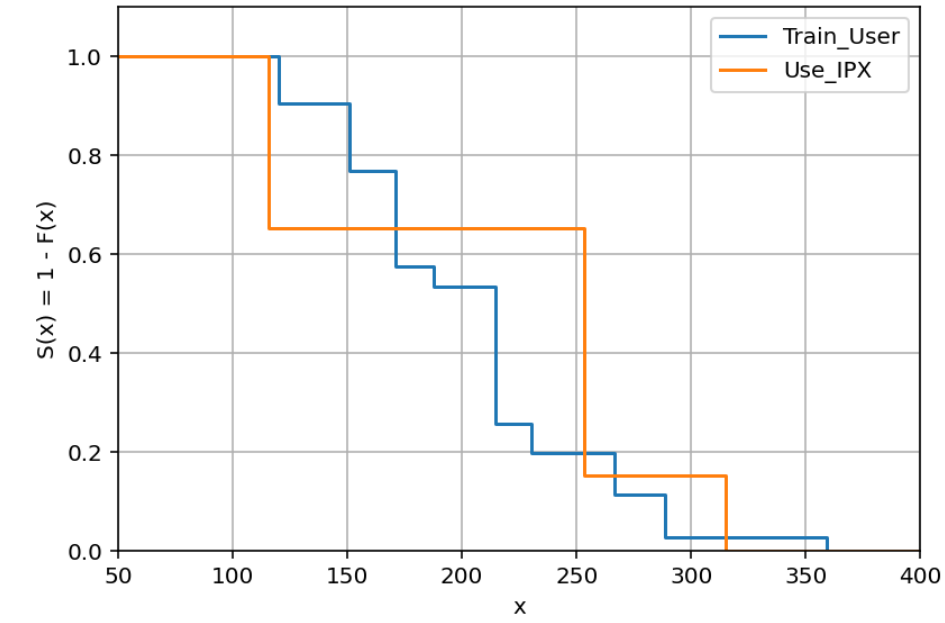
```
Train_User does not 1SD Use_IPX
```

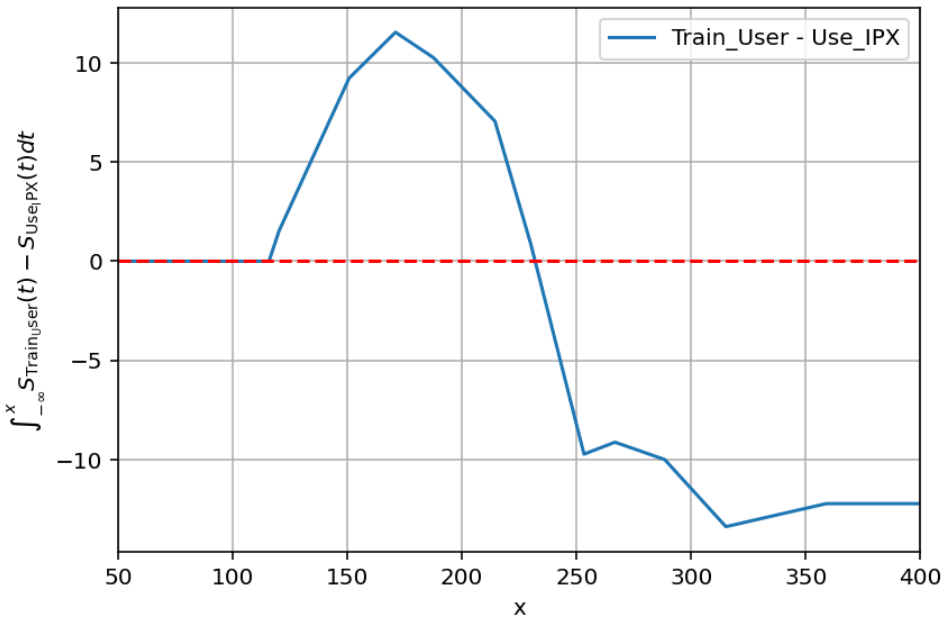


Determine if Train User 2nd order dominates Use IPX.

```
DiscreteDeal.second_order_SD(Train, IPX, xlim=xlim, plot=True)
```

```
Train_User does not 2SD Use_IPX
```





9. AHP Matrix Evaluation

9.1 Pairwise Comparison Matrix Computation

```
# Compute AHP matrix basics
from DApy.mcdm import AHPMatrix

PCM1 = [[1, 1/3, 1/2],
        [ 3,  1,   3 ],
        [ 2, 1/3,  1 ]]

# Create the AHP matrix
A1 = AHPMatrix(PCM1, labels=['Quality','Cost','Sustainability'])

# Display the matrix
A1.matrix()
```

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

Compute A1 using 'eigen' method. Available methods are 'eigen' (default), 'algebra','power', 'rgm', and 'column'. Note that 'rgm' and 'column' are approximate methods (not re-comended).

```
w, lam, ci, cr = A1.compute(method='eigen')
print(f"lambda_max={lam:.6f},  CI={ci:6f},  CR={cr:6f}")
```

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

Display matrix and all results for easy copy and paste to Word.

```
A1.print_results()
```

```
                Quality  Cost  Sustainability    Weight
Quality             1     1/3        1/2        0.157056
Cost                3      1          3         0.593634
Sustainability      2     1/3         1         0.249311

λmax=3.053622  CI=0.026811  CR=0.046225 < 0.1
```

9.2 Preference Ordering Graph

We can print the preference ordering graph for the pairwise comprison matrix. This is useful for transitivity of preference ordering violations in the matrix.

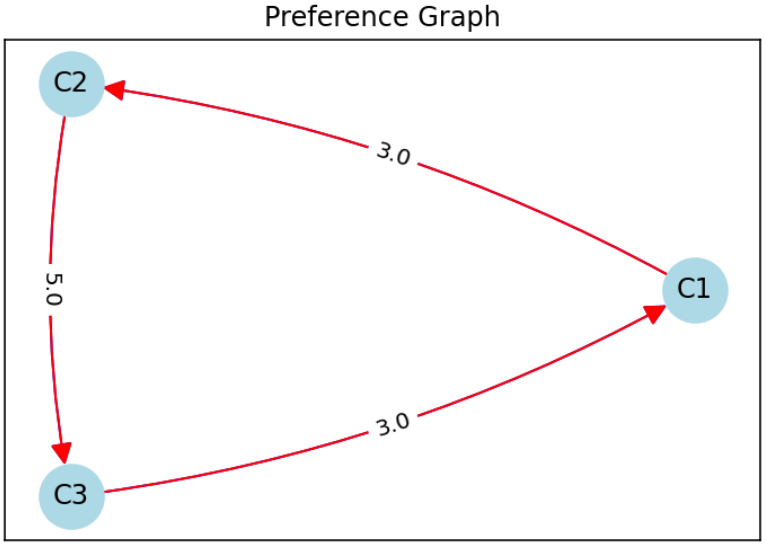
```
from DApy.mcdm import AHPMatrix

# Matrix with transivity violation
PCM2 = [[ 1,  3, 1/3],
        [1/3, 1,  5 ],
        [ 3, 1/5, 1 ]]

A2 = AHPMatrix(PCM2)
A2.compute()
A2.print_results()
```

```
     C1   C2   C3    Weight
C1   1    3   1/3  0.330135
C2  1/3   1    5   0.391418
C3   3   1/5   1   0.278447

λmax=4.838038  CI=0.919019  CR=1.584515 > 0.1
```

We note that CR=1.584515 » 0.1. Check the preference ordering graph for cycles.

```
A2.preference_graph()
```

```
1 cycles found
C3 -> C1 -> C2 -> C3
```



10. AHP Modeling and Analysis

10.1 The Job Selection Problem

We first define the criterion nodes using `mcdm.AHPNode` and then adding them into the `mcdm.AHPModel`.

```
# Job Selection Problem
from DApy.mcdm import AHPNode, AHPModel
import numpy as np

goal_pcm = np.array([
    [ 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 ]
])

goal = AHPNode("Goal", pcm=goal_pcm)

C1 = AHPNode('Research')
C2 = AHPNode('Growth')
C3 = AHPNode('Benefits')
C4 = AHPNode('Colleagues')
C5 = AHPNode('Location')
C6 = AHPNode('Reputation')

goal.children = [C1,C2,C3,C4,C5,C6]
```

Define the alternatives and their pairwise comprison matrices.

```
alternatives = ['Company A','Company B','ompany C']

# Alternative PCMs
C1.alt_pcm = np.array([
    [1, 1/4, 1/2],
    [4,  1,   3 ],
    [2, 1/3,  1 ]
])

C2.alt_pcm = np.array([
    [1, 1/4, 1/5],
```

```
    [4,  1,  1/2],
    [5,  2,   1 ]
])

C3.alt_pcm = np.array([
    [ 1,  3, 1/3],
    [1/3, 1, 1/7],
    [ 3,  7,  1 ]
])

C4.alt_pcm = np.array([
    [ 1,  1/3, 5],
    [ 3,   1,  7],
    [1/5, 1/7, 1]
])

C5.alt_pcm = np.array([
    [ 1,   1,  7],
    [ 1,   1,  7],
    [1/7, 1/7, 1]
])

C6.alt_pcm = np.array([
    [ 1,   7,  9],
    [1/7,  1,  2],
    [1/9, 1/2, 1]
])
```

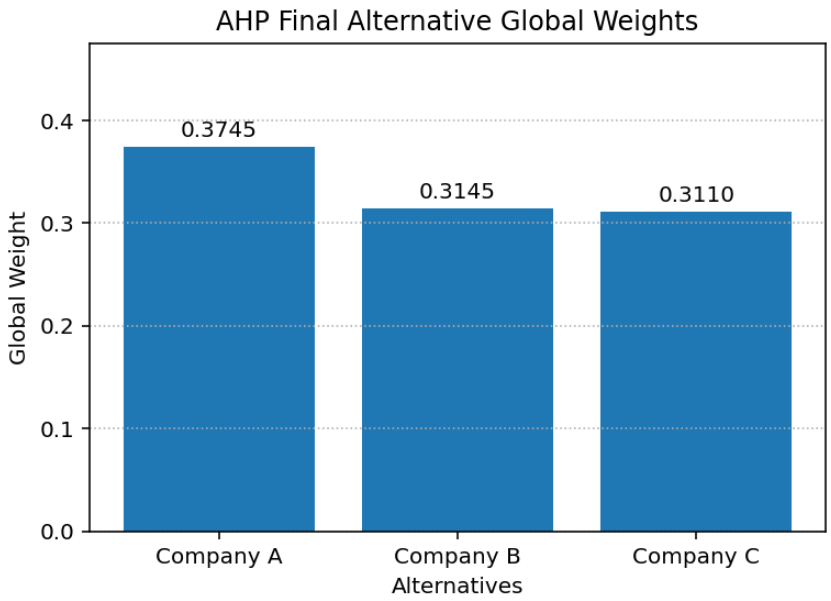
Build the AHP model and solve it.

```
model = AHPModel(goal,alternatives)
model.solve()
```

```
Alternative     Global weight
  Company A       0.374467
  Company B       0.314491
  Company C       0.311042
```

Plot the global weights bar chart.

```
model.plot_global_weights()
```



Plot the AHP Hierarchy Tree in text format.

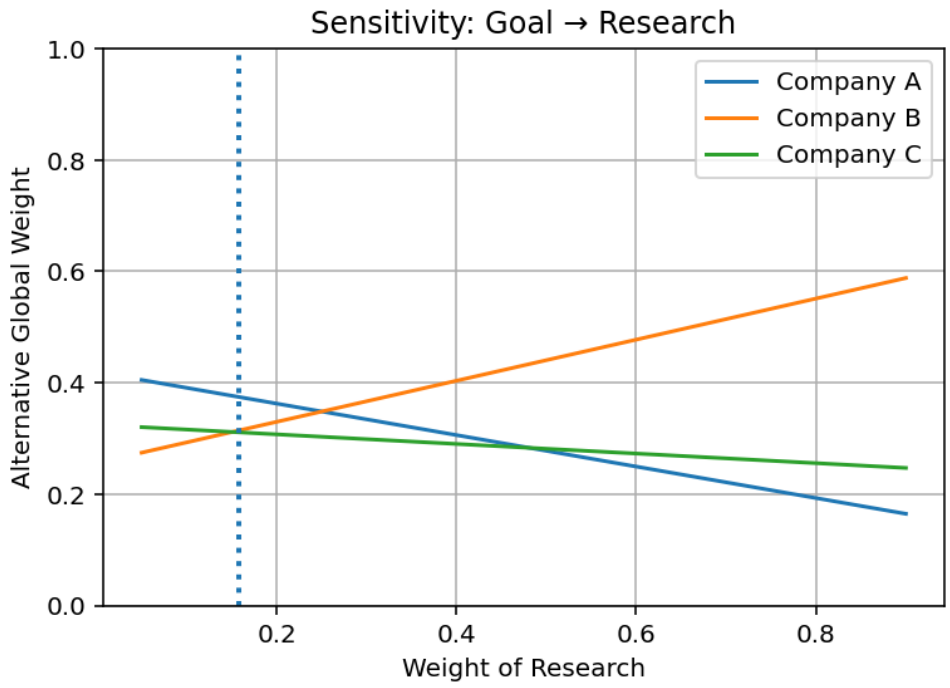
```
model.plot_text_tree()
```

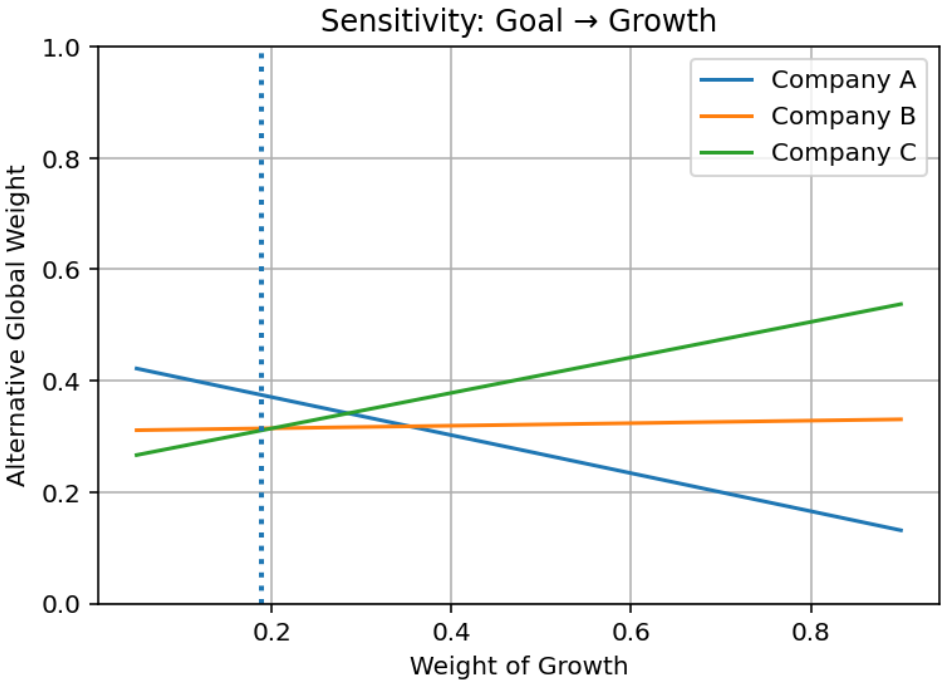
```
Goal
├── Research (0.158408)
│   Research
│   ├── Company A (0.136500)
│   ├── Company B (0.625013)
│   ├── Company C (0.238487)
├── Growth (0.189247)
│   Growth
│   ├── Company A (0.097390)
│   ├── Company B (0.333069)
│   ├── Company C (0.569541)
├── Benefits (0.197997)
│   Benefits
│   ├── Company A (0.242637)
│   ├── Company B (0.087946)
│   ├── Company C (0.669417)
├── Colleagues (0.048310)
│   Colleagues
│   ├── Company A (0.278955)
```

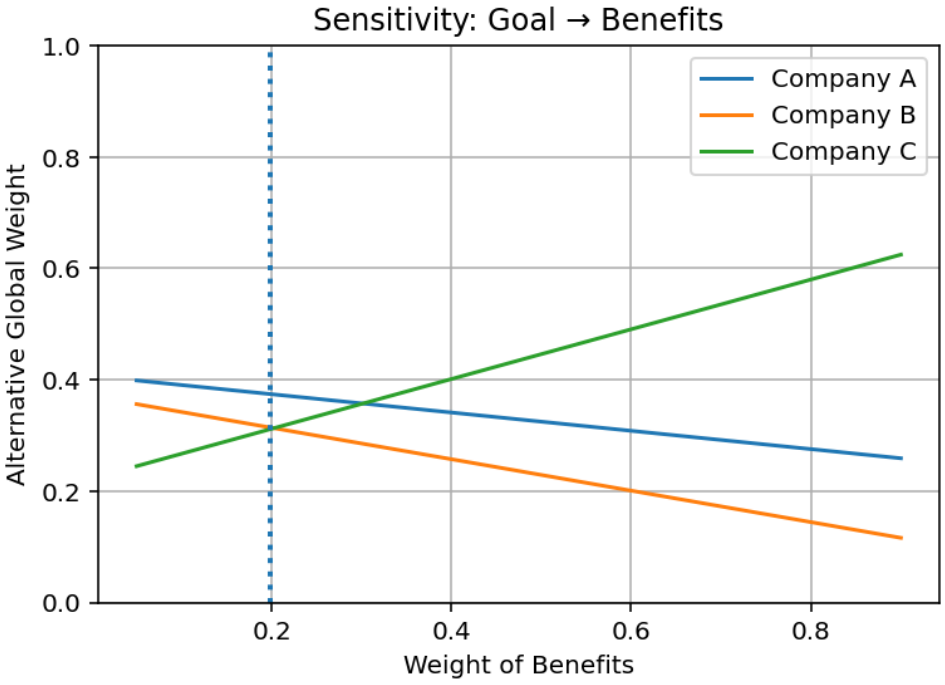
```
│    ├── Company B (0.649118)
│    ├── Company C (0.071927)
├── Location (0.150245)
│    Location
│    ├── Company A (0.466667)
│    ├── Company B (0.466667)
│    ├── Company C (0.066667)
└── Reputation (0.255792)
    Reputation
    ├── Company A (0.792757)
    ├── Company B (0.131221)
    ├── Company C (0.076021)
```

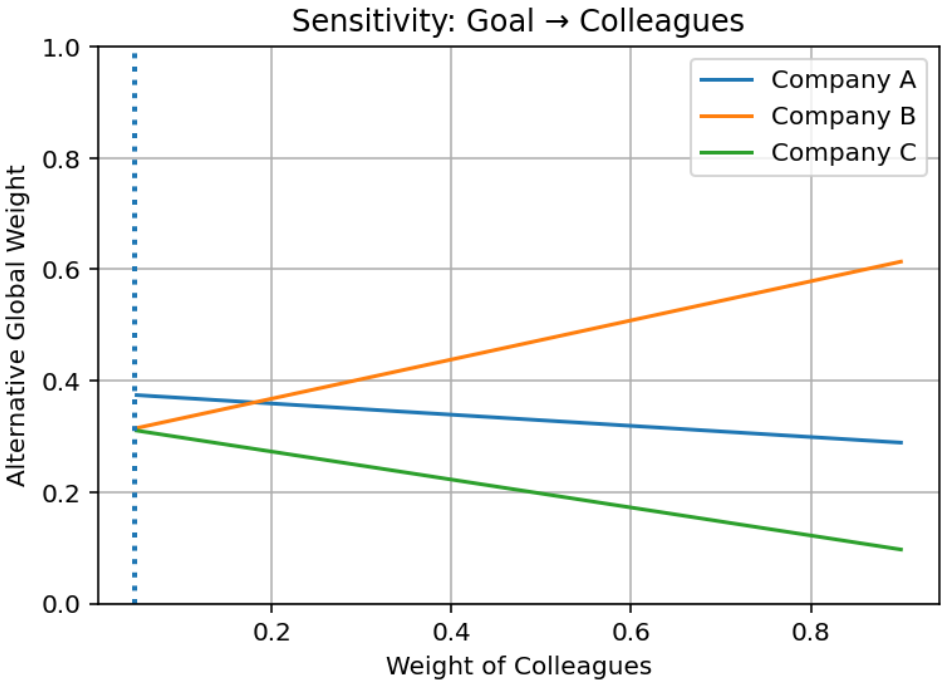
Perform sensitivity analysis on all the criteria weights.

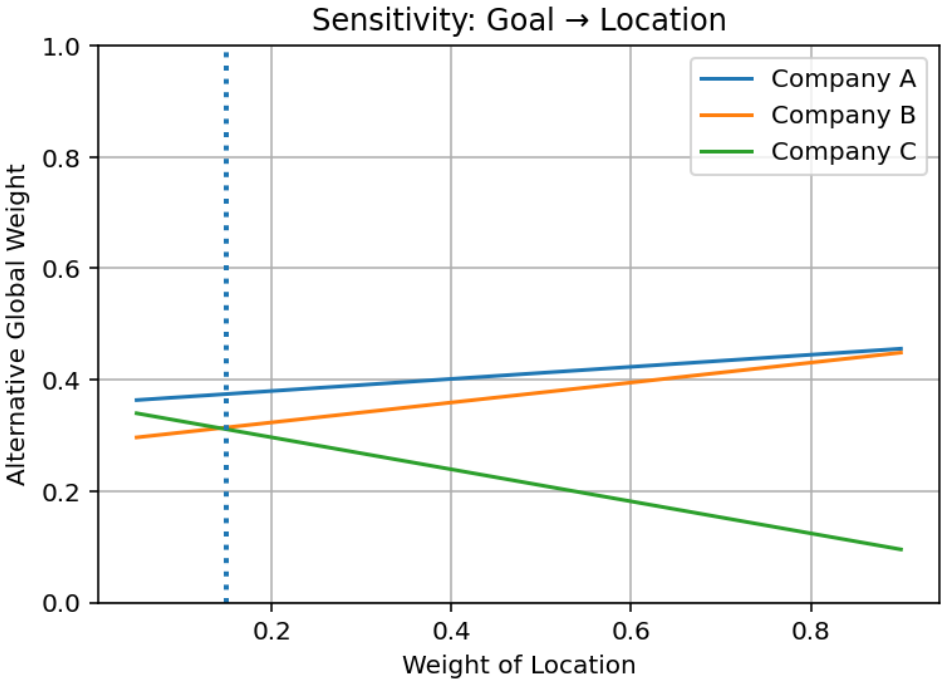
```
model.sensit()
```

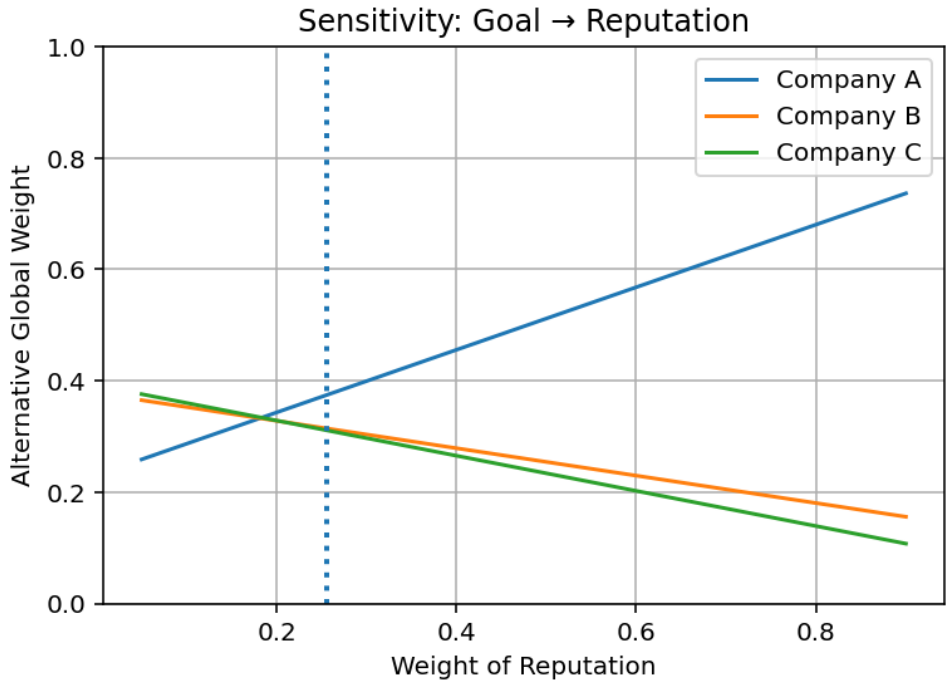



Sensitivity: Goal → Growth
Company A
Company B
Company C
Alternative Global Weight
Weight of Growth
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8


Sensitivity: Goal → Benefits
Company A
Company B
Company C
Alternative Global Weight
Weight of Benefits
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8


Sensitivity: Goal → Colleagues
Company A
Company B
Company C
Alternative Global Weight
Weight of Colleagues
0.0
0.2
0.4
0.6
0.8
1.0


Sensitivity: Goal → Location
Company A
Company B
Company C
Alternative Global Weight
Weight of Location
0.0
0.2
0.4
0.6
0.8
1.0



Display the model.

```
model.display_model()
```

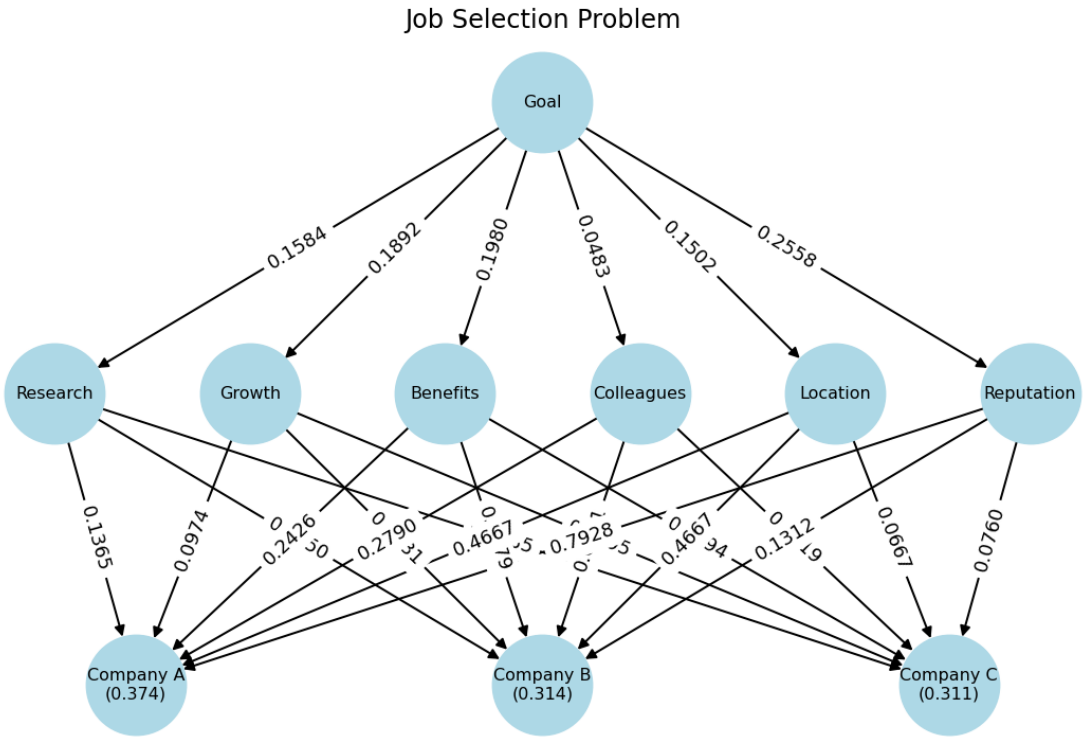
```
==================================================
Node: Goal
Pairwise Comparison Matrix
            Research  Growth  Benefits  Colleagues  Location  Reputation
Research        1.00    1.00  1.000000         4.0  1.000000    0.500000
Growth          1.00    1.00  2.000000         4.0  1.000000    0.500000
Benefits        1.00    0.50  1.000000         5.0  3.000000    0.500000
Colleagues      0.25    0.25  0.200000         1.0  0.333333    0.333333
Location        1.00    1.00  0.333333         3.0  1.000000    1.000000
Reputation      2.00    2.00  2.000000         3.0  1.000000    1.000000

Local Weights
Research      0.158408
Growth        0.189247
Benefits      0.197997
Colleagues    0.048310
Location      0.150245
Reputation    0.255792
dtype: float64

CI = 0.084069
CR = 0.067797

--------------------------------------------------
```

```
Alternatives under Research
           Company A  Company B  Company C
Company A        1.0   0.250000        0.5
Company B        4.0   1.000000        3.0
Company C        2.0   0.333333        1.0

Alternative Priorities
Company A    0.136500
Company B    0.625013
Company C    0.238487
dtype: float64

--------------------------------------------------
Alternatives under Growth
           Company A  Company B  Company C
Company A        1.0       0.25        0.2
Company B        4.0       1.00        0.5
Company C        5.0       2.00        1.0

Alternative Priorities
Company A    0.097390
Company B    0.333069
Company C    0.569541
dtype: float64

--------------------------------------------------
Alternatives under Benefits
           Company A  Company B  Company C
Company A   1.000000        3.0   0.333333
Company B   0.333333        1.0   0.142857
Company C   3.000000        7.0   1.000000

Alternative Priorities
Company A    0.242637
Company B    0.087946
Company C    0.669417
dtype: float64

--------------------------------------------------
Alternatives under Colleagues
           Company A  Company B  Company C
Company A        1.0   0.333333        5.0
Company B        3.0   1.000000        7.0
Company C        0.2   0.142857        1.0

Alternative Priorities
Company A    0.278955
Company B    0.649118
Company C    0.071927
dtype: float64
```

```
------------------------------------------------
Alternatives under Location
           Company A  Company B  Company C
Company A   1.000000   1.000000        7.0
Company B   1.000000   1.000000        7.0
Company C   0.142857   0.142857        1.0

Alternative Priorities
Company A    0.466667
Company B    0.466667
Company C    0.066667
dtype: float64

------------------------------------------------
Alternatives under Reputation
           Company A  Company B  Company C
Company A   1.000000        7.0        9.0
Company B   0.142857        1.0        2.0
Company C   0.111111        0.5        1.0

Alternative Priorities
Company A    0.792757
Company B    0.131221
Company C    0.076021
dtype: float64

================================================
Alternatives' Global Weights
================================================
Company A    0.374467
Company B    0.314491
Company C    0.311042
dtype: float64
```

Plot the AHP Hierarchy in graphic form.

```
model.plot_hierarchy(title='Job Selection Problem')
```



10.2 The Job Selection Problem with Growth Subcriteria

```
# Job Selection Problem with Growth subcriteria
from DApy.mcdm import AHPNode, AHPModel
import numpy as np

# Goal PCM (6 criteria)
goal_pcm = np.array([
    [ 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 ]
])

goal = AHPNode("Goal", pcm=goal_pcm)

C1 = AHPNode('Research')
C2 = AHPNode('Growth')
C3 = AHPNode('Benefits')
C4 = AHPNode('Colleagues')
C5 = AHPNode('Location')
C6 = AHPNode('Reputation')

goal.children=[C1,C2,C3,C4,C5,C6]

# Subcriteria for C2 'Growth'
C21 = AHPNode('Short-term')
C22 = AHPNode('Long-term')
C2.children = [C21,C22]
C2.pcm = np.array([
    [1, 1/3],
    [3,  1]
])
```

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

# Alternative PCM w.r.t. leaf criteria
C1.alt_pcm = np.array([
    [1, 1/4, 1/2],
    [4,  1,   3 ],
    [2, 1/3,  1 ]
])
```

```
C21.alt_pcm = np.array([
    [1, 1/3, 1/7],
    [3,  1,  1/3],
    [7,  3,   1 ]
])

C22.alt_pcm = np.array([
    [ 1,   3,  5],
    [1/3,  1,  2],
    [1/5, 1/2, 1]
])


C3.alt_pcm = np.array([
    [ 1,  3, 1/3],
    [1/3, 1, 1/7],
    [ 3,  7,  1 ]
])

C4.alt_pcm = np.array([
    [ 1,  1/3, 5],
    [ 3,   1,  7],
    [1/5, 1/7, 1]
])

C5.alt_pcm = np.array([
    [ 1,   1,  7],
    [ 1,   1,  7],
    [1/7, 1/7, 1]
])

C6.alt_pcm = np.array([
    [ 1,   7,  9],
    [1/7,  1,  2],
    [1/9, 1/2, 1]
])
```

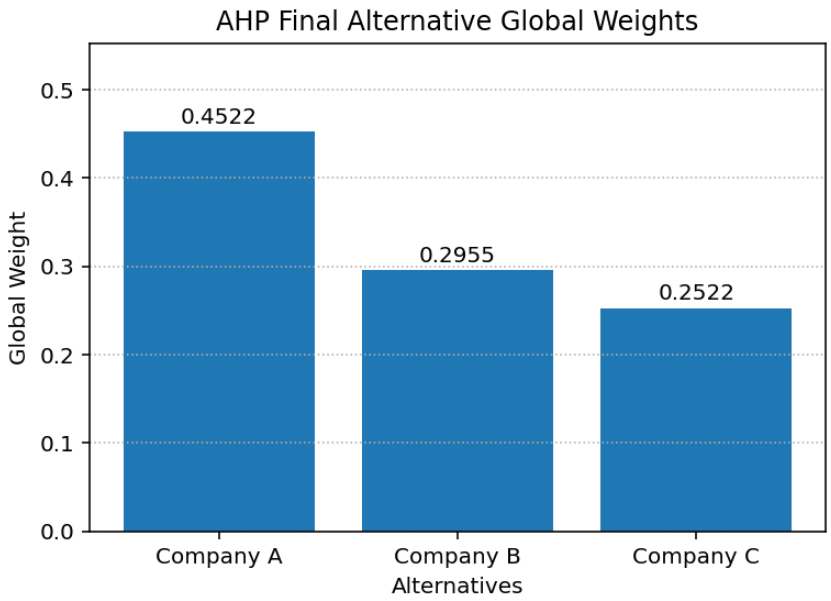
Build and solve the model.

```
model = AHPModel(goal, alternatives)
model.solve()
```

```
Alternative     Global weight
  Company A       0.452218
  Company B       0.295534
  Company C       0.252248
```

Plot the goal weights bar chart.

```
model.plot_global_weights()
```



Plot the AHP hierarchy tree in text form.

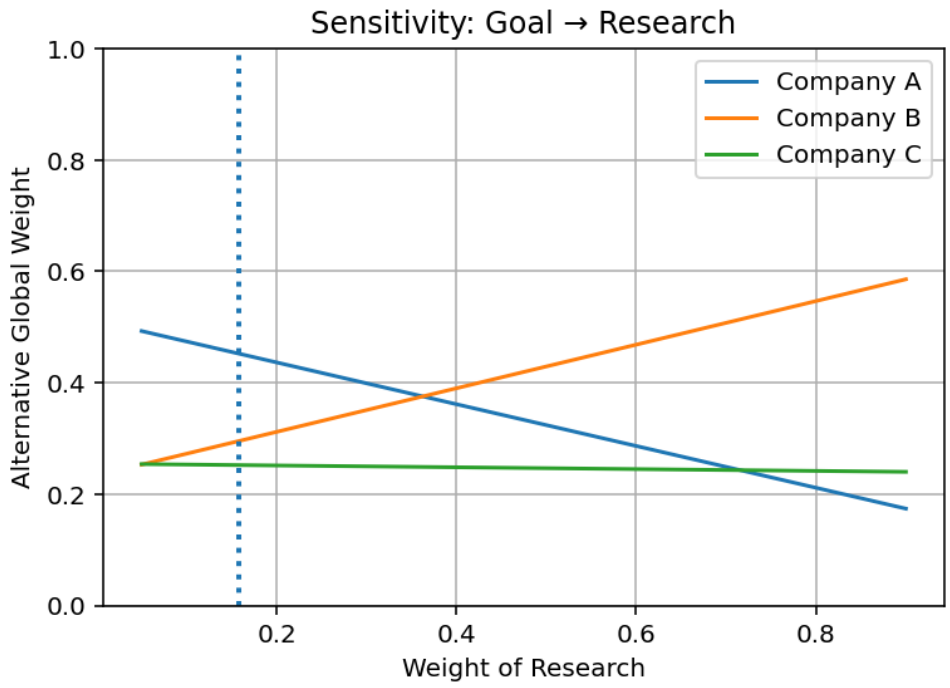
```
model.plot_text_tree()
```

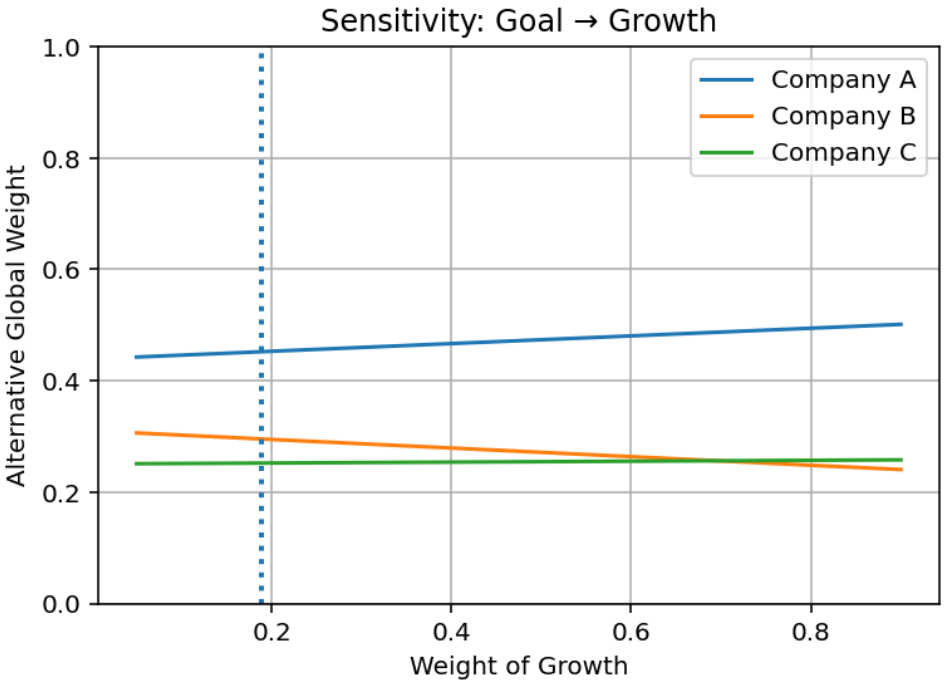
```
Goal
├── Research (0.158408)
│   Research
│   ├── Company A (0.136500)
│   ├── Company B (0.625013)
│   ├── Company C (0.238487)
├── Growth (0.189247)
│   Growth
│   ├── Short-term (0.250000)
│   │   Short-term
│   │   ├── Company A (0.087946)
│   │   ├── Company B (0.242637)
│   │   ├── Company C (0.669417)
│   └── Long-term (0.750000)
│       Long-term
│       ├── Company A (0.648329)
│       ├── Company B (0.229651)
│       ├── Company C (0.122020)
├── Benefits (0.197997)
```

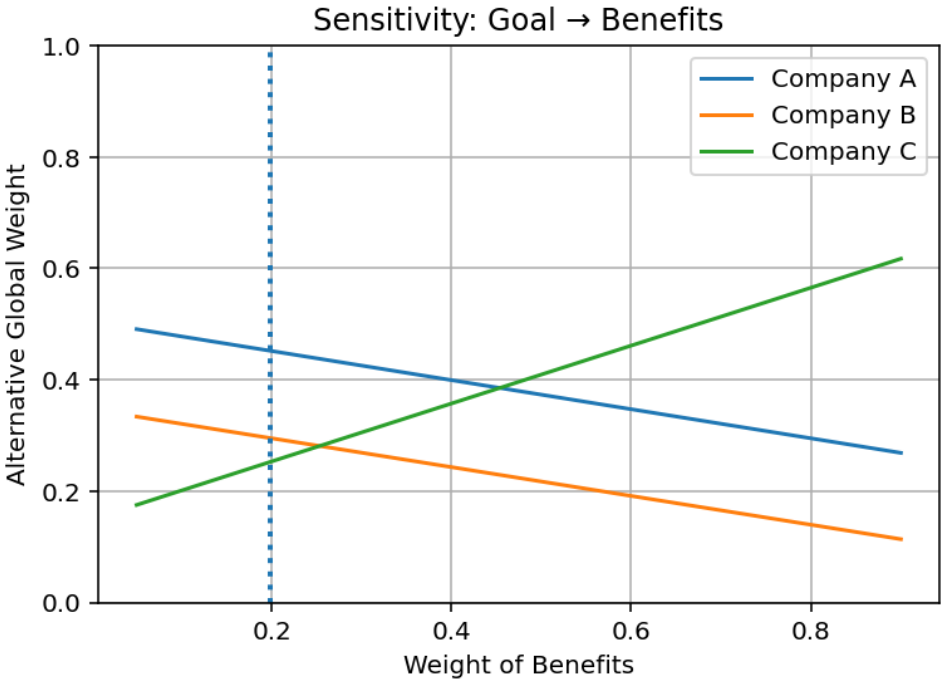
```
│    Benefits
│    ├── Company A (0.242637)
│    ├── Company B (0.087946)
│    ├── Company C (0.669417)
├── Colleagues (0.048310)
│    Colleagues
│    ├── Company A (0.278955)
│    ├── Company B (0.649118)
│    ├── Company C (0.071927)
├── Location (0.150245)
│    Location
│    ├── Company A (0.466667)
│    ├── Company B (0.466667)
│    ├── Company C (0.066667)
└── Reputation (0.255792)
     Reputation
     ├── Company A (0.792757)
     ├── Company B (0.131221)
     ├── Company C (0.076021)
```

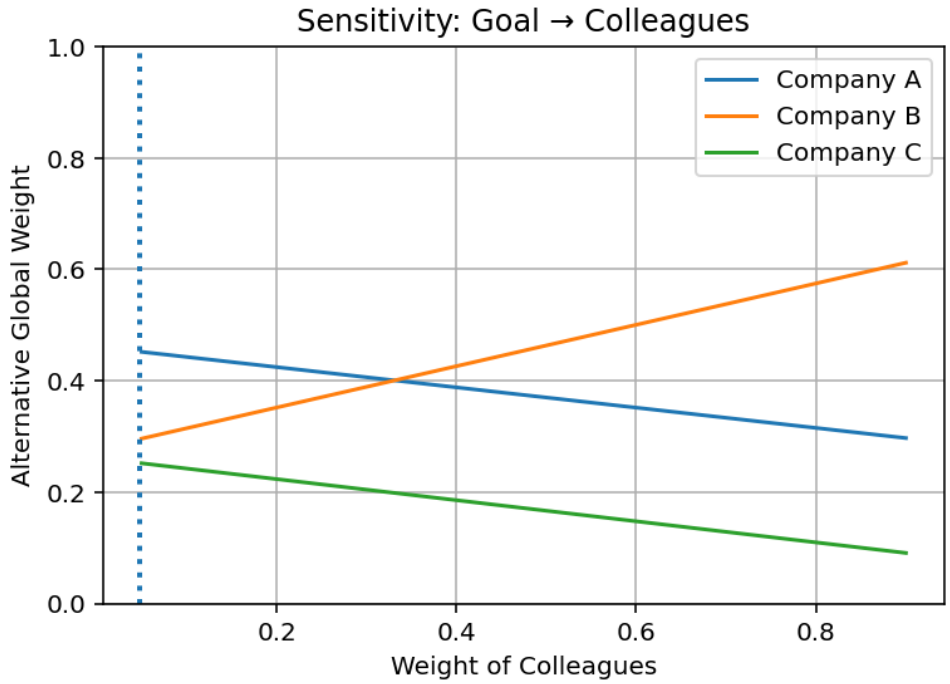
Perform sensivity analysis on all criteria weights.

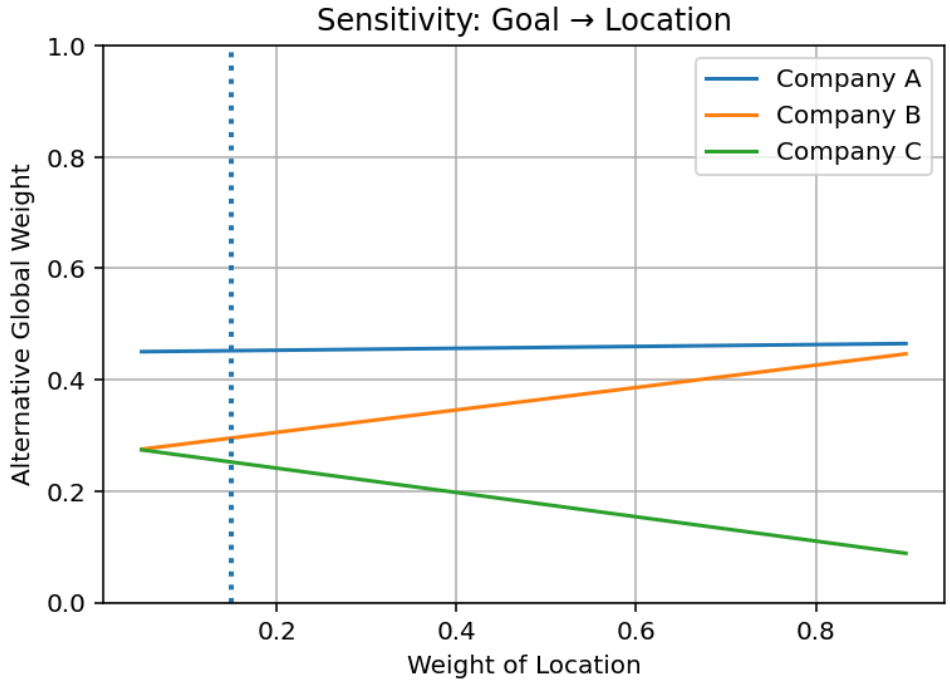
```
model.sensit()
```

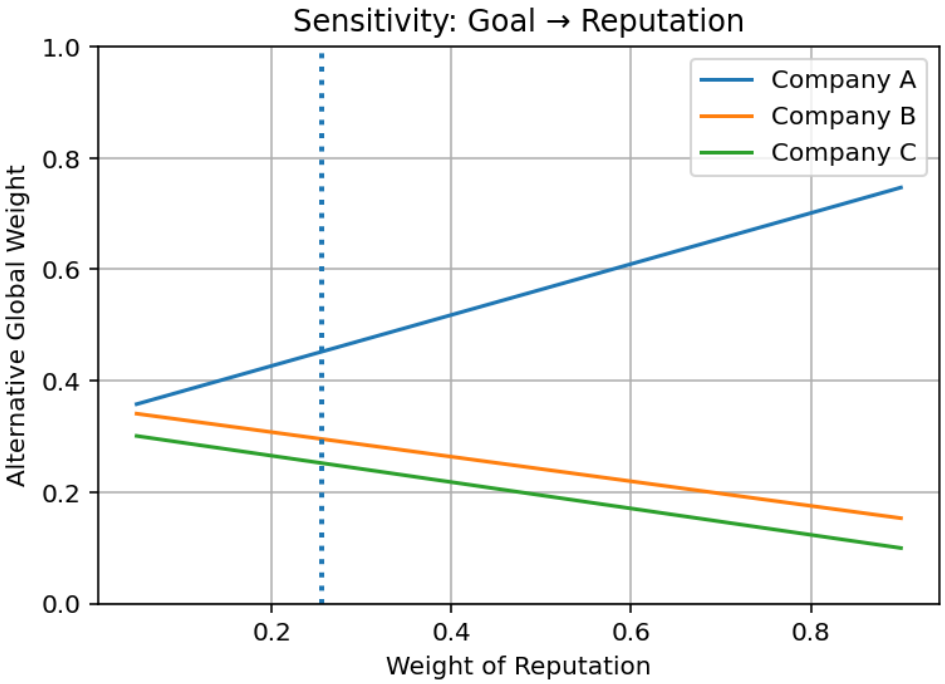


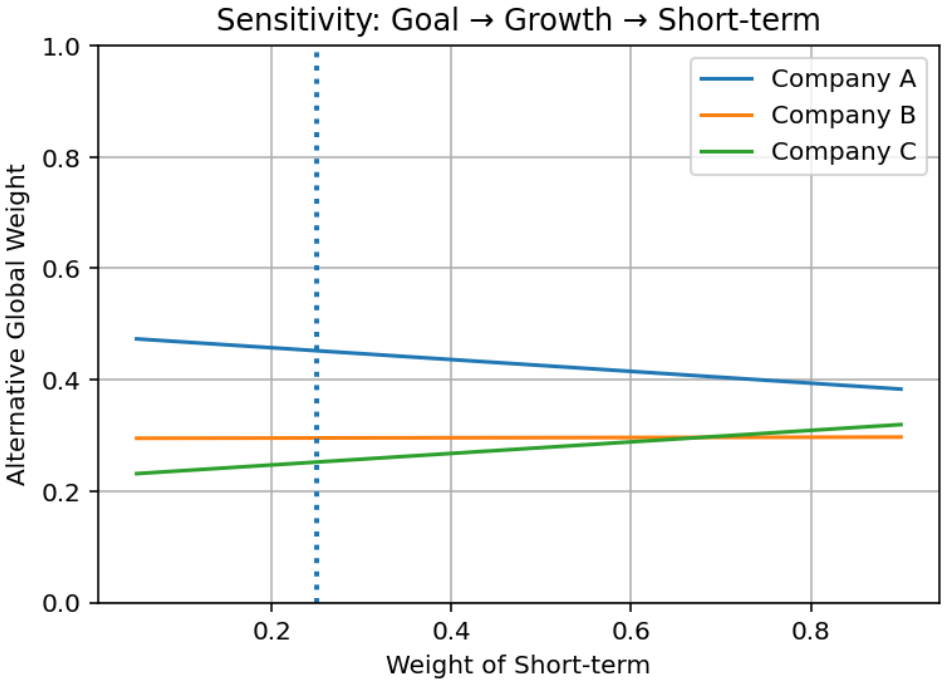

Sensitivity: Goal → Growth
Company A
Company B
Company C
Alternative Global Weight
Weight of Growth
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8

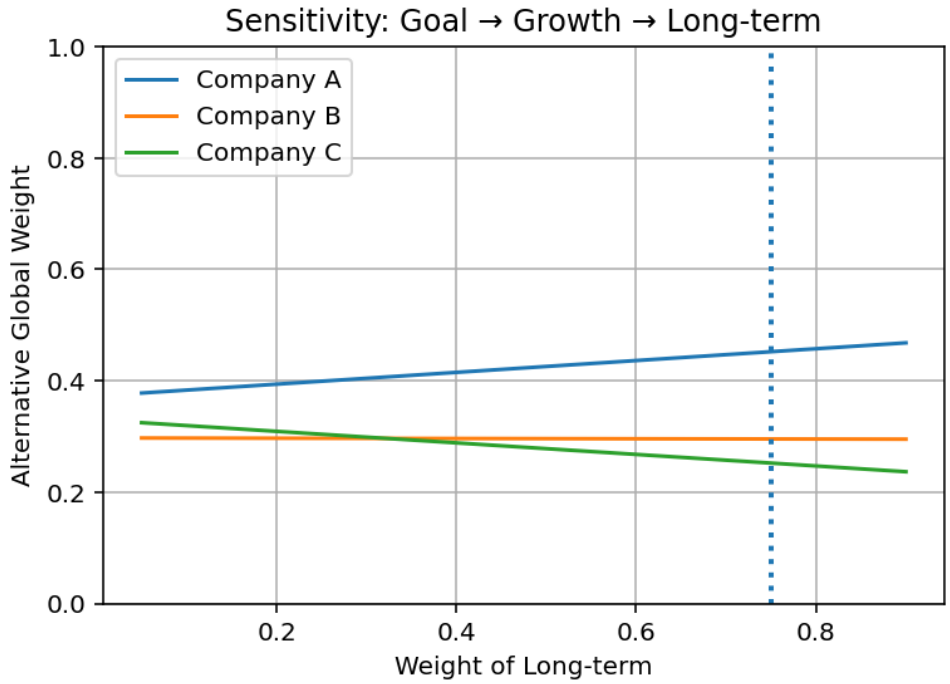

Sensitivity: Goal → Benefits
Company A
Company B
Company C
Alternative Global Weight
Weight of Benefits
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8


Sensitivity: Goal → Colleagues
Company A
Company B
Company C
Alternative Global Weight
Weight of Colleagues
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8


Sensitivity: Goal → Location
Company A
Company B
Company C
Alternative Global Weight
Weight of Location
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8


Sensitivity: Goal → Reputation
Company A
Company B
Company C
Alternative Global Weight
Weight of Reputation
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8


Sensitivity: Goal → Growth → Short-term
Company A
Company B
Company C
Alternative Global Weight
Weight of Short-term
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8



Show all the model information and details.

```
model.display_model()
```

```
================================================
Node: Goal
Pairwise Comparison Matrix
            Research  Growth  Benefits  Colleagues  Location  Reputation
Research        1.00    1.00  1.000000         4.0  1.000000    0.500000
Growth          1.00    1.00  2.000000         4.0  1.000000    0.500000
Benefits        1.00    0.50  1.000000         5.0  3.000000    0.500000
Colleagues      0.25    0.25  0.200000         1.0  0.333333    0.333333
Location        1.00    1.00  0.333333         3.0  1.000000    1.000000
Reputation      2.00    2.00  2.000000         3.0  1.000000    1.000000

Local Weights
Research      0.158408
Growth        0.189247
Benefits      0.197997
Colleagues    0.048310
Location      0.150245
Reputation    0.255792
dtype: float64

CI = 0.084069
CR = 0.067797

------------------------------------------------
```

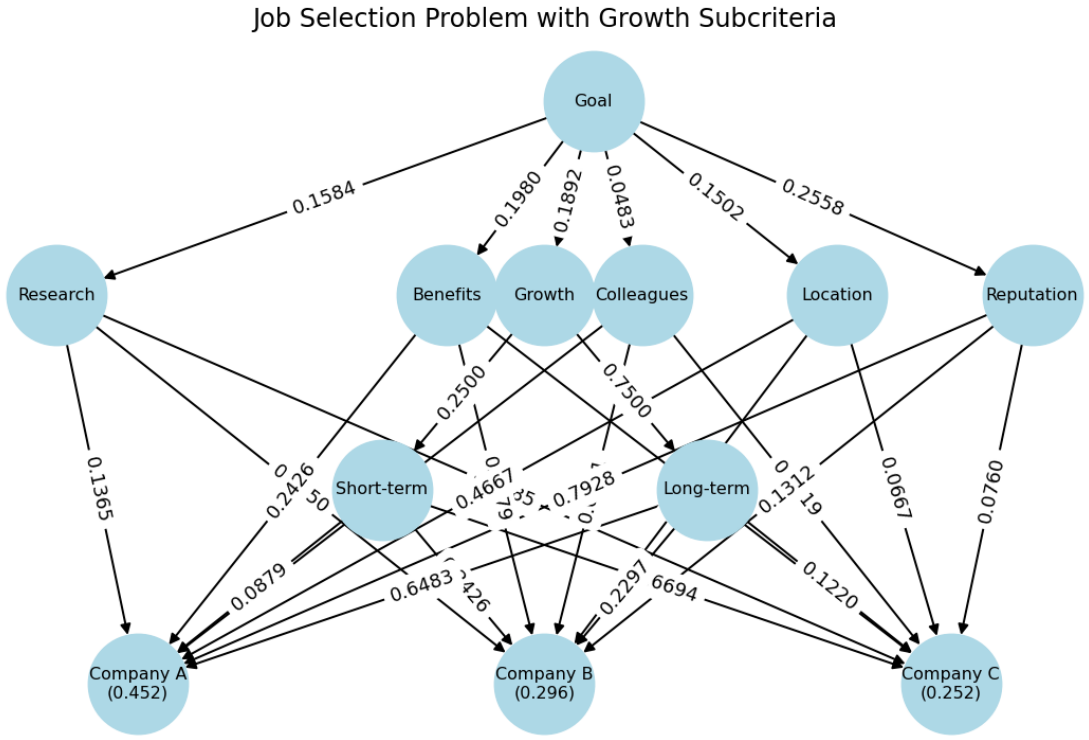
```
Alternatives under Research
           Company A  Company B  Company C
Company A        1.0   0.250000        0.5
Company B        4.0   1.000000        3.0
Company C        2.0   0.333333        1.0

Alternative Priorities
Company A    0.136500
Company B    0.625013
Company C    0.238487
dtype: float64

================================================
Node: Growth
Pairwise Comparison Matrix
            Short-term  Long-term
Short-term         1.0   0.333333
Long-term          3.0   1.000000

Local Weights
Short-term    0.25
Long-term     0.75
dtype: float64

CI = 0.000000
CR = 0.000000

------------------------------------------------
Alternatives under Short-term
           Company A  Company B  Company C
Company A        1.0   0.333333   0.142857
Company B        3.0   1.000000   0.333333
Company C        7.0   3.000000   1.000000

Alternative Priorities
Company A    0.087946
Company B    0.242637
Company C    0.669417
dtype: float64

------------------------------------------------
Alternatives under Long-term
           Company A  Company B  Company C
Company A   1.000000        3.0        5.0
Company B   0.333333        1.0        2.0
Company C   0.200000        0.5        1.0

Alternative Priorities
Company A    0.648329
Company B    0.229651
Company C    0.122020
```

```
dtype: float64

------------------------------------------------
Alternatives under Benefits
           Company A  Company B  Company C
Company A   1.000000        3.0   0.333333
Company B   0.333333        1.0   0.142857
Company C   3.000000        7.0   1.000000

Alternative Priorities
Company A    0.242637
Company B    0.087946
Company C    0.669417
dtype: float64

------------------------------------------------
Alternatives under Colleagues
           Company A  Company B  Company C
Company A        1.0   0.333333        5.0
Company B        3.0   1.000000        7.0
Company C        0.2   0.142857        1.0

Alternative Priorities
Company A    0.278955
Company B    0.649118
Company C    0.071927
dtype: float64

------------------------------------------------
Alternatives under Location
           Company A  Company B  Company C
Company A   1.000000   1.000000        7.0
Company B   1.000000   1.000000        7.0
Company C   0.142857   0.142857        1.0

Alternative Priorities
Company A    0.466667
Company B    0.466667
Company C    0.066667
dtype: float64

------------------------------------------------
Alternatives under Reputation
           Company A  Company B  Company C
Company A   1.000000        7.0        9.0
Company B   0.142857        1.0        2.0
Company C   0.111111        0.5        1.0

Alternative Priorities
Company A    0.792757
Company B    0.131221
```

```
Company C    0.076021
dtype: float64

================================================
Alternatives' Global Weights
================================================
Company A    0.452218
Company B    0.295534
Company C    0.252248
dtype: float64
```

Plot the AHP model in graphic form.

```
model.plot_hierarchy(title='Job Selection Problem with Growth
   Subcriteria')
```



10.3 The Substainable Energy System Problem

```
# Sustainable Energy System Problem (2026 04 14)
from DApy.mcdm import AHPNode, AHPModel
import numpy as np

goal_pcm = np.array([
    [1,  1,  1/5, 1/2],
    [1,  1,  1/5, 1/2],
    [5,  5,   1,   3 ],
    [2,  2,  1/3,  1 ]
])

goal = AHPNode("Goal", pcm=goal_pcm)

C1 = AHPNode('Technical')
C2 = AHPNode('Economic')
C3 = AHPNode('Environmental')
C4 = AHPNode('Social')

goal.children = [C1,C2,C3,C4]

# Subcriteria for C1 'Technical'
C11 = AHPNode('Tech readiness')
C12 = AHPNode('Safety')
C13 = AHPNode('Efficiency')
C14 = AHPNode('Useful life')
C1.children = [C11,C12,C13,C14]
C1.pcm = np.array([
    [ 1,   1,  3,  1 ],
    [ 1,   1,  3,  1 ],
    [1/3, 1/3, 1, 1/2],
    [ 1,   1,  2,  1 ]
])

# Subcriteria for C2 'Economic'
C21 = AHPNode('Investment cost')
C22 = AHPNode('O&M costs')
C23 = AHPNode('Feed-in-tariff')
C2.children = [C21,C22,C23]
C2.pcm = np.array([
    [1, 1/9, 1/5],
    [9,  1,   4 ],
    [5, 1/4,  1 ]
])

# Subcriteria for C3  'Environmental'
C31 = AHPNode('CO2 emission')
C32 = AHPNode('Land use')
C33 = AHPNode('Water consumption')
```

```
C3.children = [C31,C32,C33]
C3.pcm = np.array([
    [1, 1/4, 1/3],
    [4,  1,   1 ],
    [3,  1,   1 ]
])

# Subcriteria for C4  'Social'
C41 = AHPNode('Jobs creation')
C42 = AHPNode('Meet demands')
C4.children = [C41,C42]
C4.pcm = np.array([
    [ 1,  2],
    [1/2, 1]
])
```

```
alternatives = ['Solar PV','Wind','Nuclear','Biomass']

# Alternative PCMs
C11.alt_pcm = np.array([
    [ 1,   1,  3,  2 ],
    [ 1,   1,  3,  2 ],
    [1/3, 1/3, 1, 1/2],
    [1/2, 1/2, 2,  1 ]
])

C12.alt_pcm = np.array([
    [ 1,   2,  3, 1/2],
    [1/2,  1,  2, 1/2],
    [1/3, 1/2, 1, 1/4],
    [ 2,   2,  4,  1 ]
])

C13.alt_pcm = np.array([
    [1, 1/2, 1/2, 1/2],
    [2,  1,   1,   2 ],
    [2,  1,   1,   1 ],
    [2,  1/2, 1,   1 ]
])

C14.alt_pcm = np.array([
    [1, 1, 1/2,  1 ],
    [1, 1, 1/3, 1/2],
    [2, 3,  1,   3 ],
    [1, 2, 1/3,  1 ]
])

C21.alt_pcm = np.array([
```

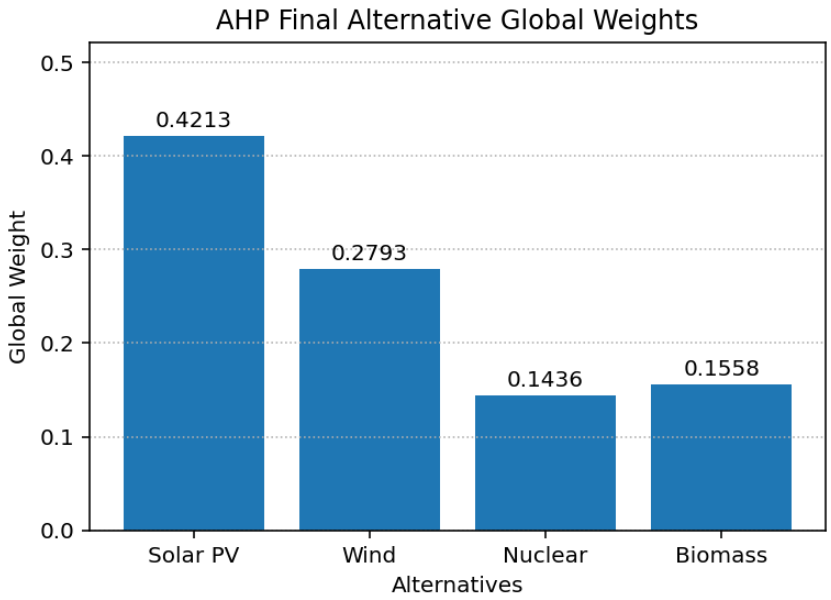
```
    [1, 1/2,  1, 1],
    [2,  1,   2, 2],
    [1,  1/2, 1, 1],
    [1,  1/2, 1, 1]
])

C22.alt_pcm = np.array([
    [ 1,  2, 1/3, 1/7],
    [1/2, 1, 1/3, 1/9],
    [ 3,  3,  1,  1/3],
    [ 7,  9,  3,   1 ]
])

C23.alt_pcm = np.array([
    [1, 1/2, 1/2, 1/2],
    [2,  1,   1,   2 ],
    [2,  1,   1,   1 ],
    [2, 1/2,  1,   1 ]
])

C31.alt_pcm = np.array([
    [ 1, 1/2,  1/2, 3],
    [ 2,  1,   1/2, 5],
    [ 2,  2,    1,  5],
    [1/3, 1/5, 1/5, 1]
])

C32.alt_pcm = np.array([
    [ 1,   8,   7,  9],
    [1/8,  1,   1,  4],
    [1/7,  1,   1,  4],
    [1/9, 1/4, 1/4, 1]
])

C33.alt_pcm = np.array([
    [ 1, 1/2,  9,  8 ],
    [ 2,  1,   9,  9 ],
    [1/9, 1/9, 1, 1/2],
    [1/8, 1/9, 2,  1 ]
])

C41.alt_pcm = np.array([
    [ 1,  5, 6,  4 ],
    [1/5, 1, 1,  1 ],
    [1/6, 1, 1, 1/2],
    [1/4, 1, 2,  1 ]
])

C42.alt_pcm = np.array([
    [ 1, 1/5,  2, 1/2],
    [ 5,  1,   7,  2 ],
```

```
    [1/2, 1/7, 1, 1/3],
    [ 2,  1/2, 3,  1 ]
])
```

```
# Build and solve the model
model = AHPModel(goal, alternatives)
model.solve()
```

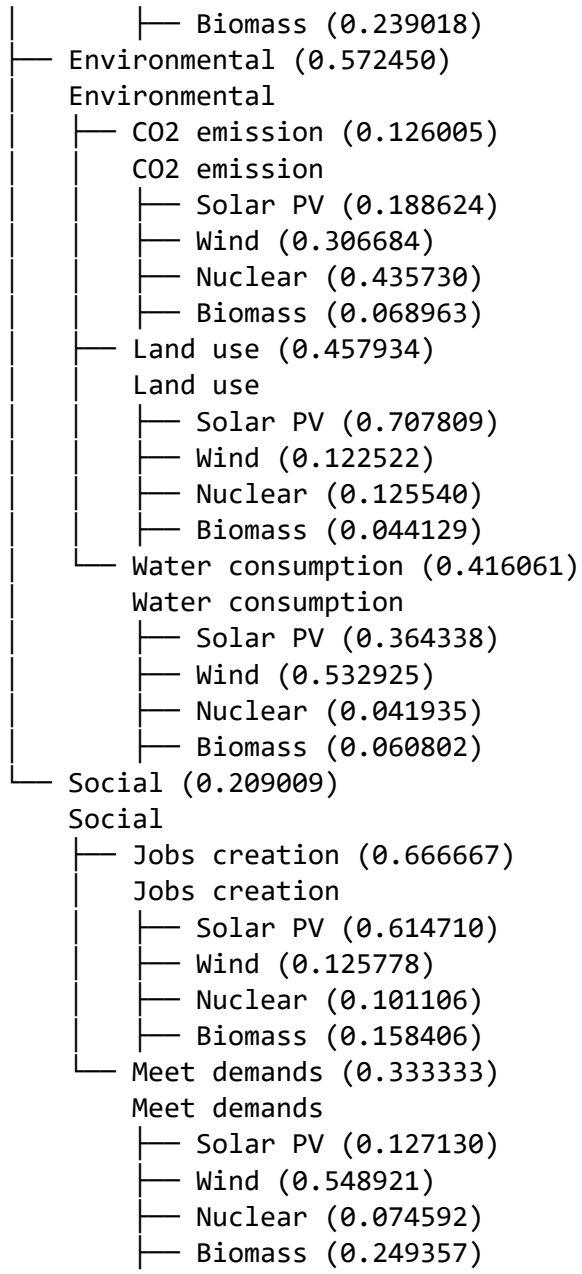
```
Alternative     Global weight
  Solar PV        0.421290
  Wind            0.279321
  Nuclear         0.143571
  Biomass         0.155818
```

```
model.plot_global_weights()
```

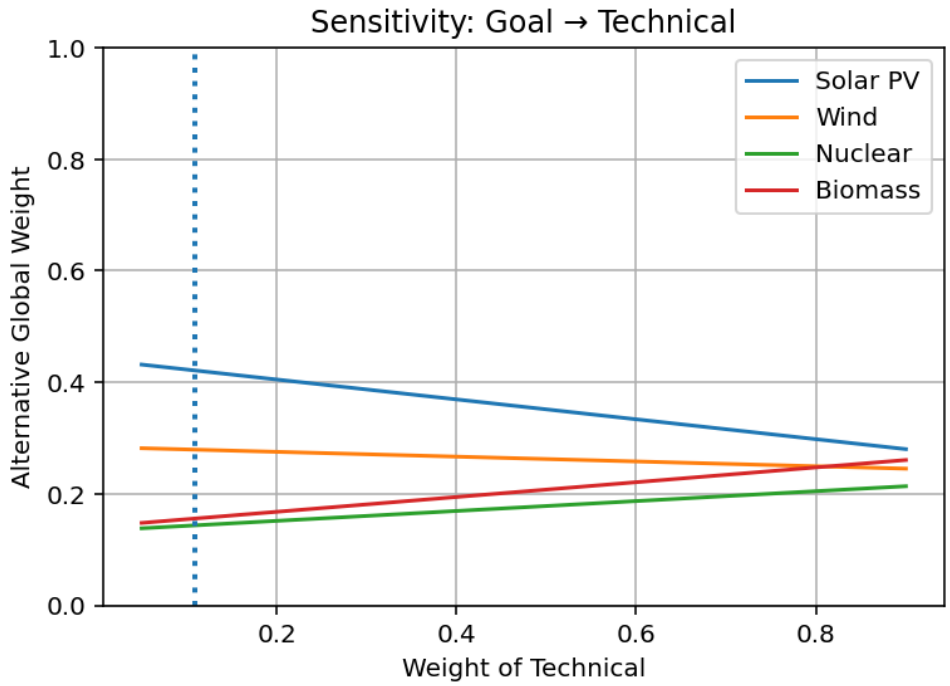


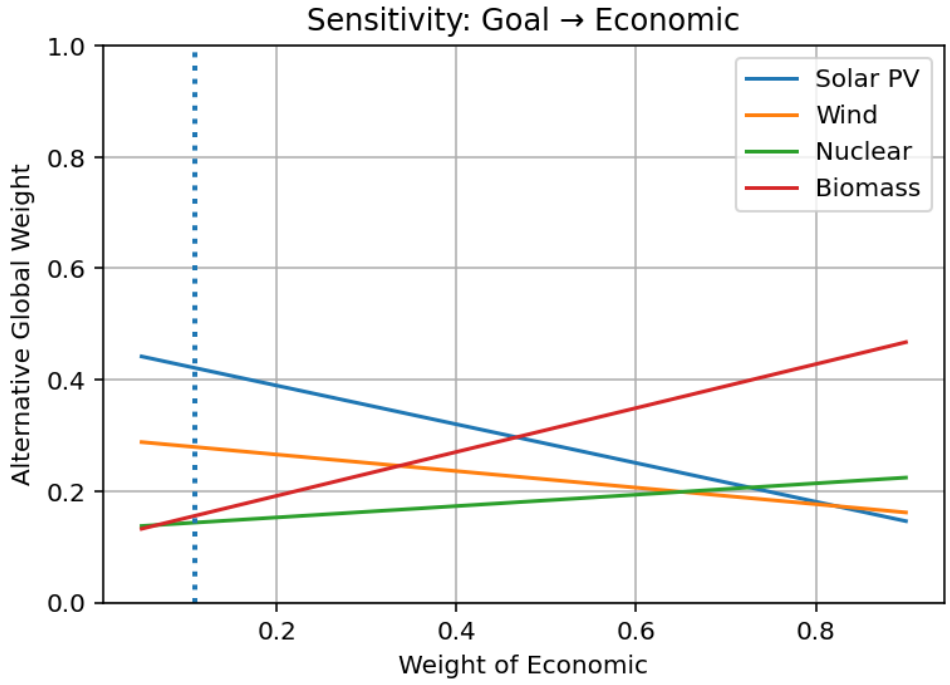
```
model.plot_text_tree()
```

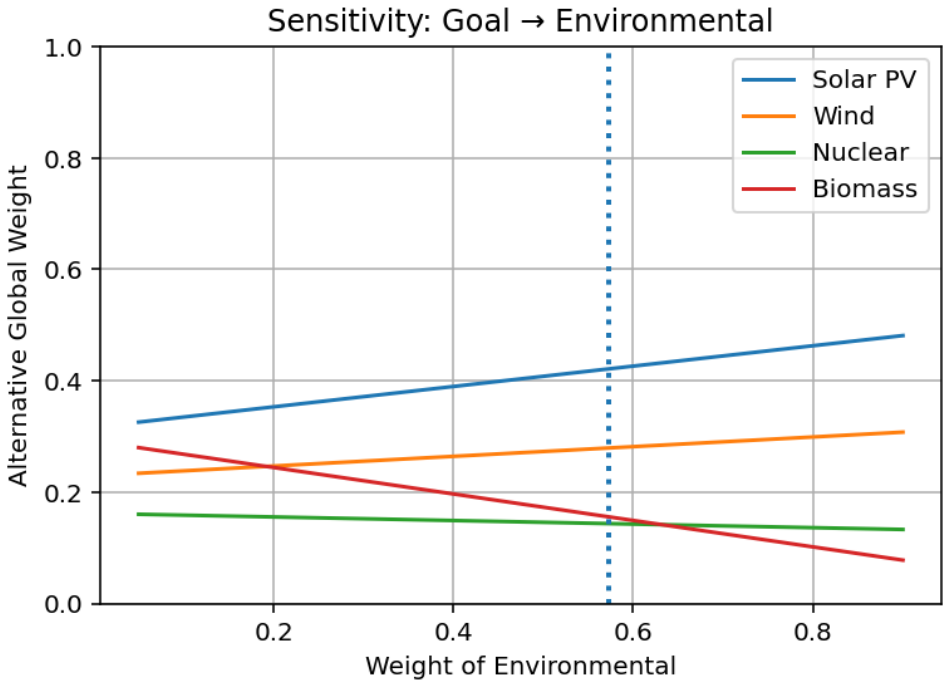
```
Goal
├── Technical (0.109270)
│   Technical
│   ├── Tech readiness (0.304999)
│   │   Tech readiness
│   │   ├── Solar PV (0.350913)
│   │   ├── Wind (0.350913)
│   │   ├── Nuclear (0.109114)
│   │   ├── Biomass (0.189060)
│   ├── Safety (0.304999)
│   │   Safety
│   │   ├── Solar PV (0.286323)
│   │   ├── Wind (0.182003)
│   │   ├── Nuclear (0.096899)
│   │   ├── Biomass (0.434775)
│   ├── Efficiency (0.113143)
│   │   Efficiency
│   │   ├── Solar PV (0.140424)
│   │   ├── Wind (0.339710)
│   │   ├── Nuclear (0.280848)
│   │   ├── Biomass (0.239018)
│   └── Useful life (0.276859)
│       Useful life
│       ├── Solar PV (0.188380)
│       ├── Wind (0.144399)
│       ├── Nuclear (0.462341)
│       ├── Biomass (0.204880)
├── Economic (0.109270)
│   Economic
│   ├── Investment cost (0.060328)
│   │   Investment cost
│   │   ├── Solar PV (0.200000)
│   │   ├── Wind (0.400000)
│   │   ├── Nuclear (0.200000)
│   │   ├── Biomass (0.200000)
│   ├── O&M costs (0.708524)
│   │   O&M costs
│   │   ├── Solar PV (0.094454)
│   │   ├── Wind (0.062475)
│   │   ├── Nuclear (0.222439)
│   │   ├── Biomass (0.620632)
│   └── Feed-in-tariff (0.231148)
│       Feed-in-tariff
│       ├── Solar PV (0.140424)
│       ├── Wind (0.339710)
│       ├── Nuclear (0.280848)
```

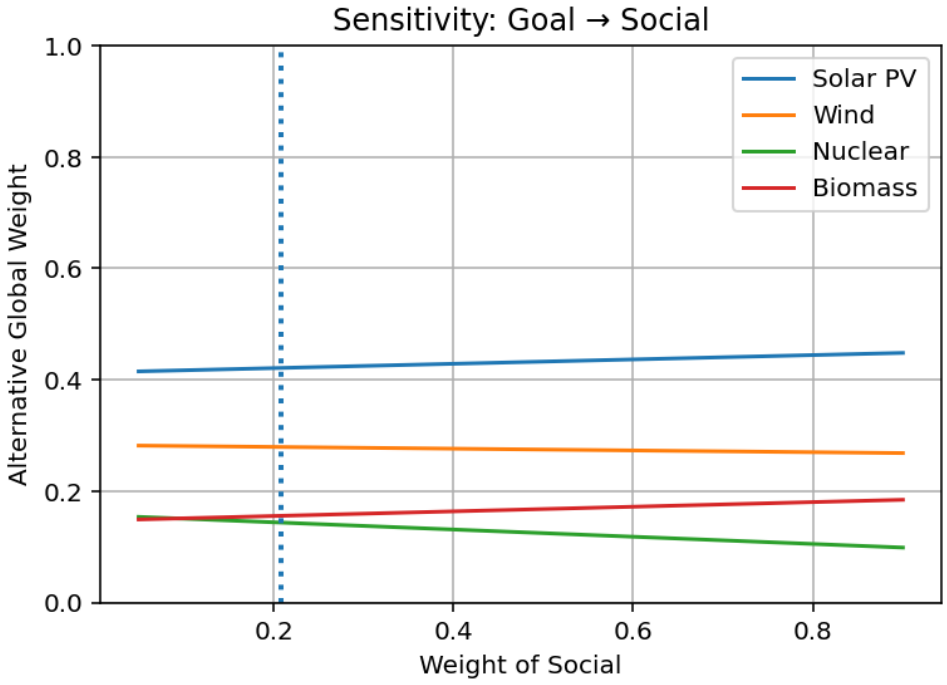

├── Biomass (0.239018)
├── Environmental (0.572450)
Environmental
├── CO2 emission (0.126005)
CO2 emission
├── Solar PV (0.188624)
├── Wind (0.306684)
├── Nuclear (0.435730)
├── Biomass (0.068963)
├── Land use (0.457934)
Land use
├── Solar PV (0.707809)
├── Wind (0.122522)
├── Nuclear (0.125540)
├── Biomass (0.044129)
└── Water consumption (0.416061)
Water consumption
├── Solar PV (0.364338)
├── Wind (0.532925)
├── Nuclear (0.041935)
├── Biomass (0.060802)
└── Social (0.209009)
Social
├── Jobs creation (0.666667)
Jobs creation
├── Solar PV (0.614710)
├── Wind (0.125778)
├── Nuclear (0.101106)
├── Biomass (0.158406)
└── Meet demands (0.333333)
Meet demands
├── Solar PV (0.127130)
├── Wind (0.548921)
├── Nuclear (0.074592)
├── Biomass (0.249357)

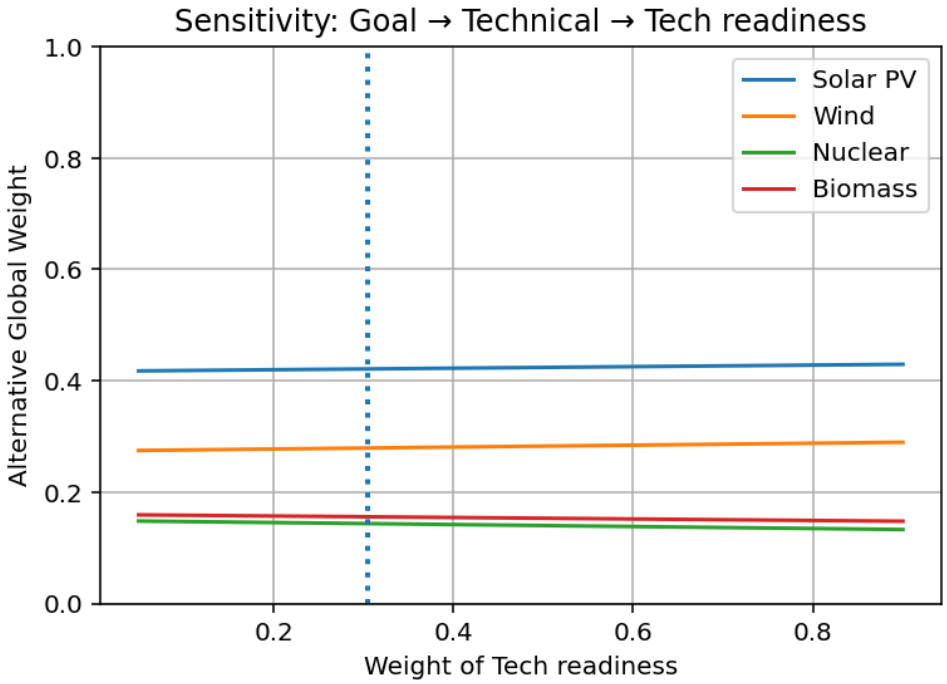
```
model.sensit()
```

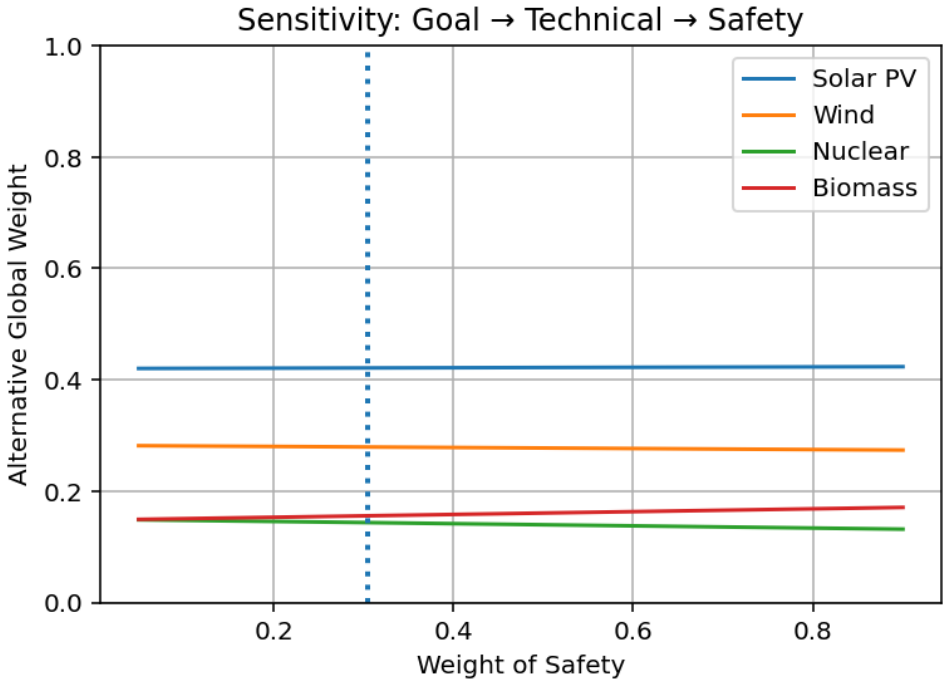


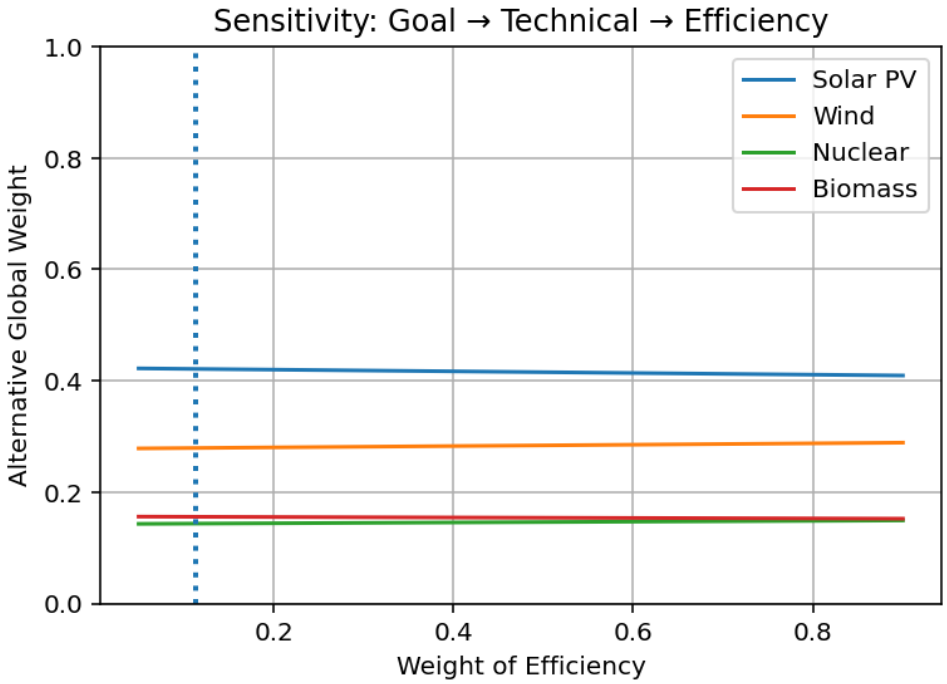


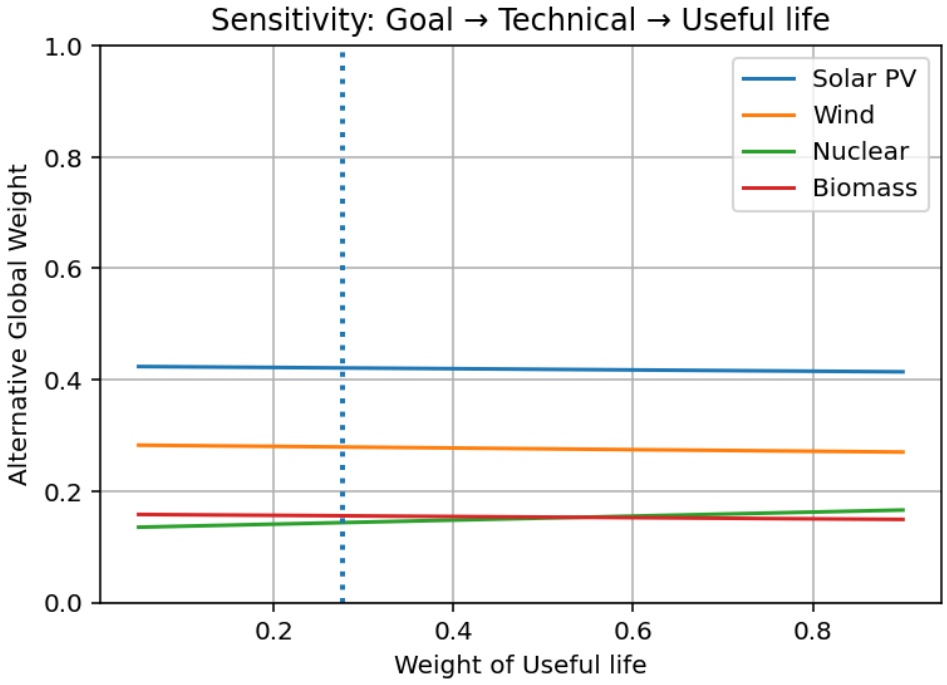

Sensitivity: Goal → Environmental
Alternative Global Weight
Weight of Environmental
Solar PV
Wind
Nuclear
Biomass
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8

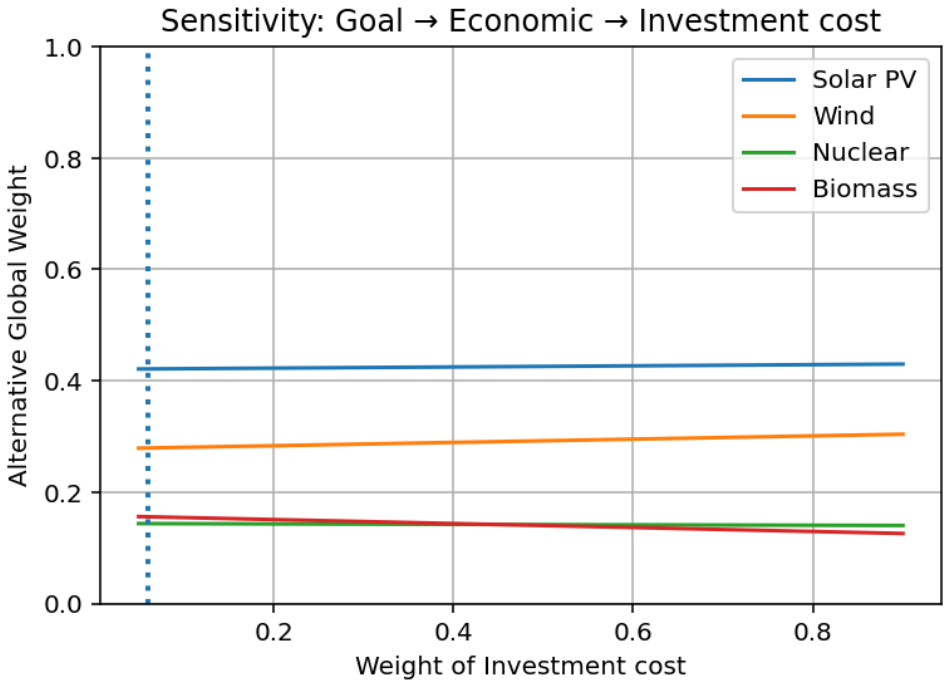

Sensitivity: Goal → Social
Alternative Global Weight
Weight of Social
Solar PV
Wind
Nuclear
Biomass
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8

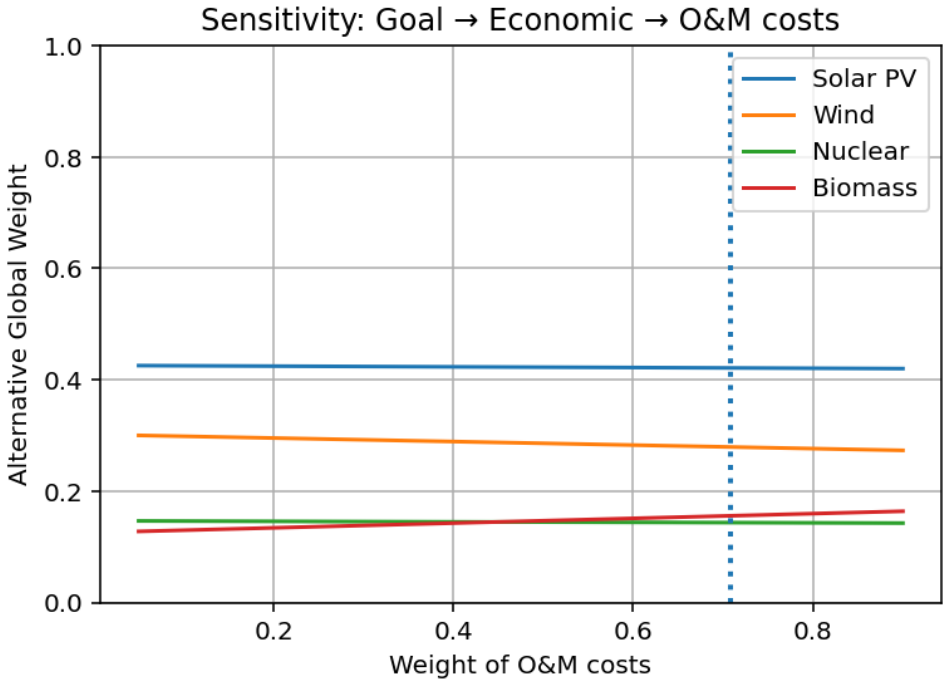

Sensitivity: Goal → Technical → Tech readiness
Solar PV
Wind
Nuclear
Biomass
Alternative Global Weight
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
Weight of Tech readiness

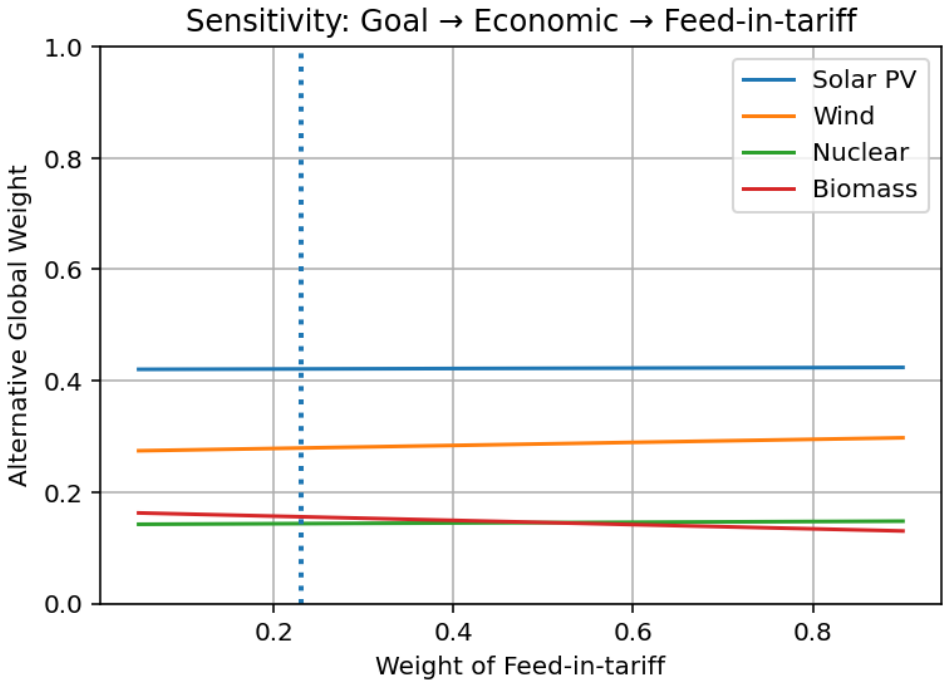

Sensitivity: Goal → Technical → Safety
Solar PV
Wind
Nuclear
Biomass
Alternative Global Weight
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
Weight of Safety

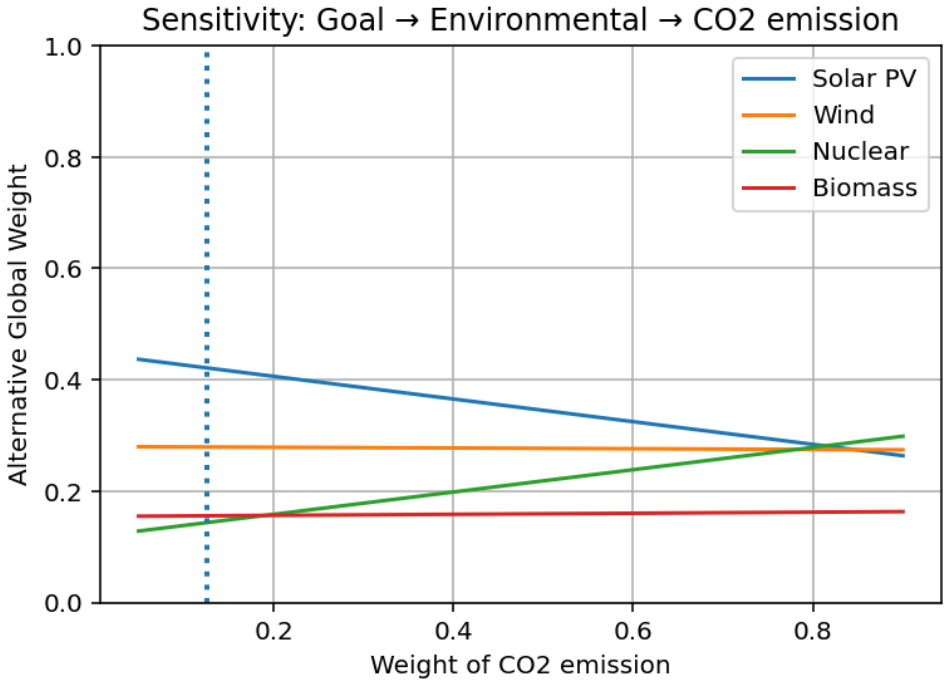

Sensitivity: Goal → Technical → Efficiency
Alternative Global Weight
Weight of Efficiency
Solar PV
Wind
Nuclear
Biomass
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8

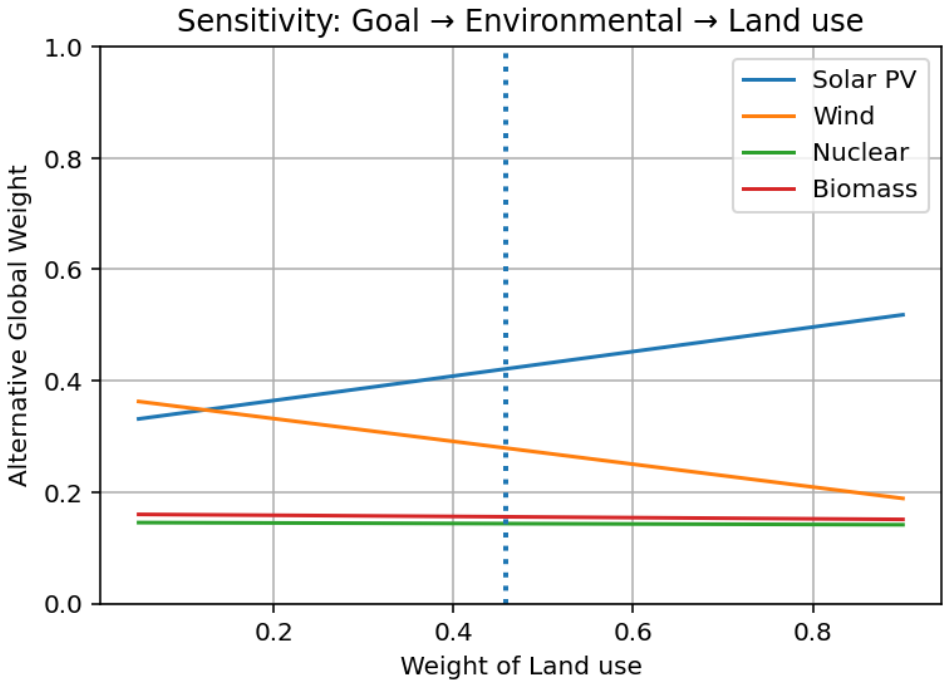

Sensitivity: Goal → Technical → Useful life
Alternative Global Weight
Weight of Useful life
Solar PV
Wind
Nuclear
Biomass
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8

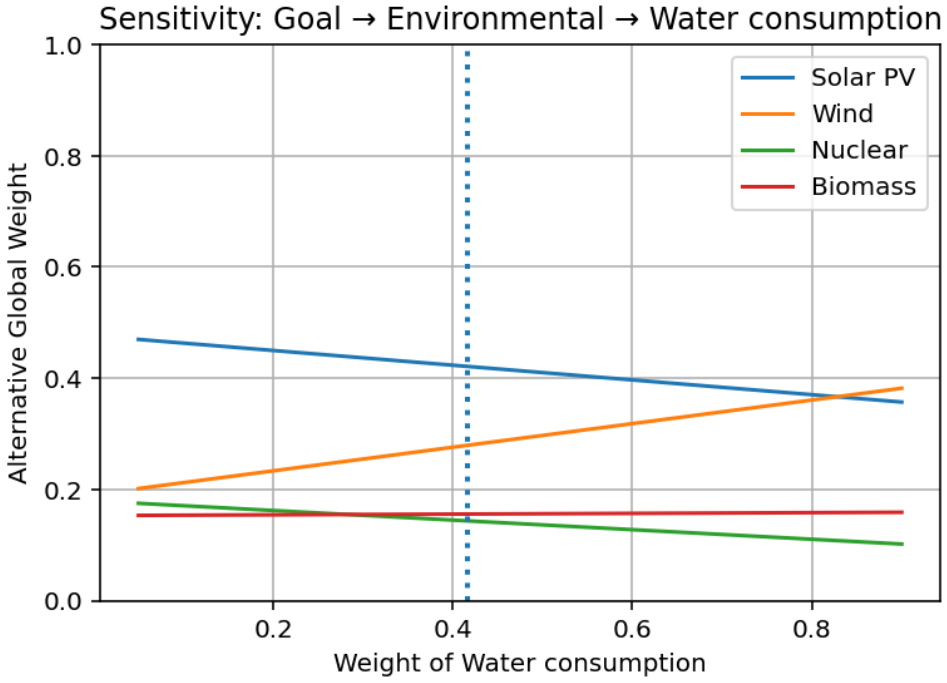

Sensitivity: Goal → Economic → Investment cost
Alternative Global Weight
Weight of Investment cost
Solar PV
Wind
Nuclear
Biomass
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8

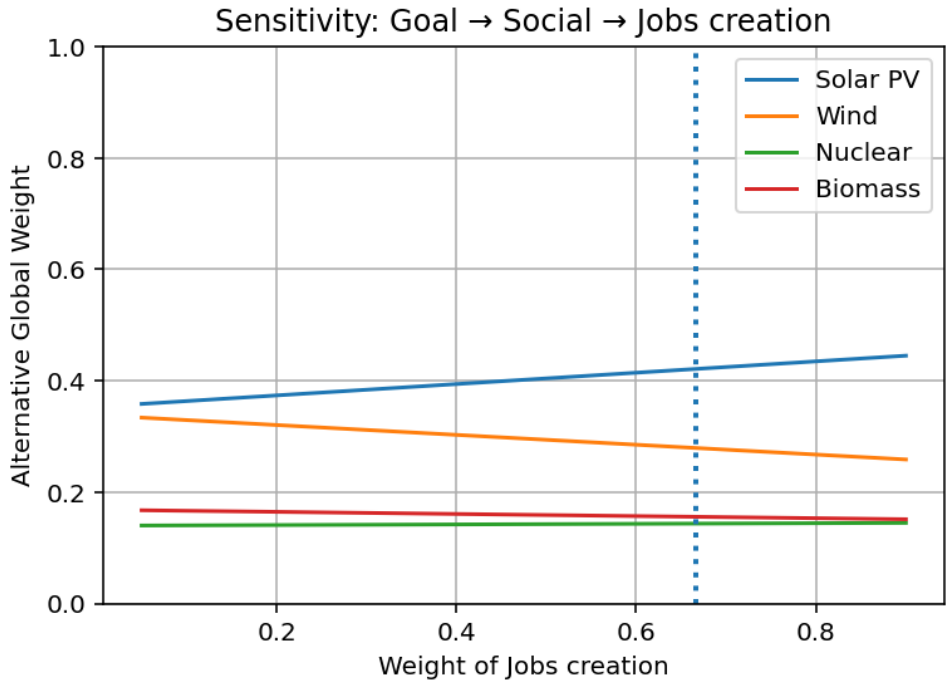

Sensitivity: Goal → Economic → O&M costs
Alternative Global Weight
Weight of O&M costs
Solar PV
Wind
Nuclear
Biomass
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8

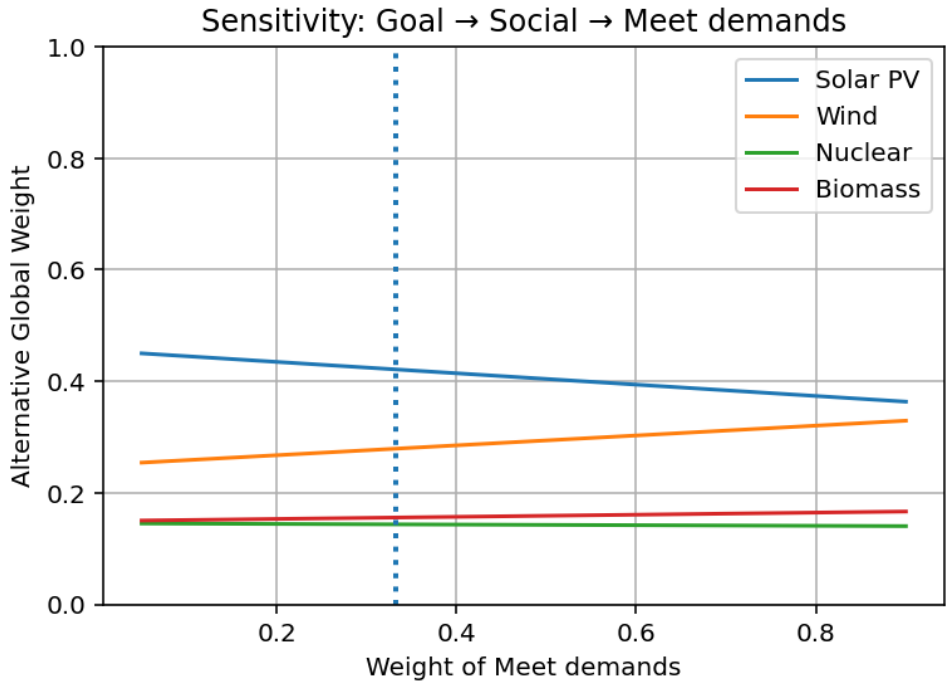

Sensitivity: Goal → Economic → Feed-in-tariff
Alternative Global Weight
Weight of Feed-in-tariff
Solar PV
Wind
Nuclear
Biomass
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8


Sensitivity: Goal → Environmental → CO2 emission
Alternative Global Weight
Weight of CO2 emission
Solar PV
Wind
Nuclear
Biomass
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8


Sensitivity: Goal → Environmental → Land use
Alternative Global Weight
Weight of Land use
Solar PV
Wind
Nuclear
Biomass
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8


Sensitivity: Goal → Environmental → Water consumption
Alternative Global Weight
Weight of Water consumption
Solar PV
Wind
Nuclear
Biomass
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8






```
model.display_model()
```

```
================================================
Node: Goal
Pairwise Comparison Matrix
               Technical  Economic  Environmental  Social
Technical            1.0       1.0       0.200000     0.5
```

```
Economic             1.0       1.0       0.200000     0.5
Environmental        5.0       5.0       1.000000     3.0
Social               2.0       2.0       0.333333     1.0

Local Weights
Technical        0.109270
Economic         0.109270
Environmental    0.572450
Social           0.209009
dtype: float64

CI = 0.001386
CR = 0.001540

================================================
Node: Technical
Pairwise Comparison Matrix
                Tech readiness    Safety  Efficiency  Useful life
Tech readiness        1.000000  1.000000         3.0          1.0
Safety                1.000000  1.000000         3.0          1.0
Efficiency            0.333333  0.333333         1.0          0.5
Useful life           1.000000  1.000000         2.0          1.0

Local Weights
Tech readiness    0.304999
Safety            0.304999
Efficiency        0.113143
Useful life       0.276859
dtype: float64

CI = 0.006873
CR = 0.007637

------------------------------------------------
Alternatives under Tech readiness
          Solar PV      Wind  Nuclear  Biomass
Solar PV  1.000000  1.000000      3.0      2.0
Wind      1.000000  1.000000      3.0      2.0
Nuclear   0.333333  0.333333      1.0      0.5
Biomass   0.500000  0.500000      2.0      1.0

Alternative Priorities
Solar PV    0.350913
Wind        0.350913
Nuclear     0.109114
Biomass     0.189060
dtype: float64

------------------------------------------------
Alternatives under Safety
          Solar PV  Wind  Nuclear  Biomass
```

```
Solar PV  1.000000   2.0      3.0     0.50
Wind      0.500000   1.0      2.0     0.50
Nuclear   0.333333   0.5      1.0     0.25
Biomass   2.000000   2.0      4.0     1.00

Alternative Priorities
Solar PV    0.286323
Wind        0.182003
Nuclear     0.096899
Biomass     0.434775
dtype: float64

--------------------------------------------------
Alternatives under Efficiency
          Solar PV  Wind  Nuclear  Biomass
Solar PV       1.0   0.5      0.5      0.5
Wind           2.0   1.0      1.0      2.0
Nuclear        2.0   1.0      1.0      1.0
Biomass        2.0   0.5      1.0      1.0

Alternative Priorities
Solar PV    0.140424
Wind        0.339710
Nuclear     0.280848
Biomass     0.239018
dtype: float64

--------------------------------------------------
Alternatives under Useful life
          Solar PV  Wind   Nuclear  Biomass
Solar PV       1.0   1.0  0.500000      1.0
Wind           1.0   1.0  0.333333      0.5
Nuclear        2.0   3.0  1.000000      3.0
Biomass        1.0   2.0  0.333333      1.0

Alternative Priorities
Solar PV    0.188380
Wind        0.144399
Nuclear     0.462341
Biomass     0.204880
dtype: float64

==================================================
Node: Economic
Pairwise Comparison Matrix
                 Investment cost  O&M costs  Feed-in-tariff
Investment cost              1.0   0.111111             0.2
O&M costs                    9.0   1.000000             4.0
Feed-in-tariff               5.0   0.250000             1.0

Local Weights
```

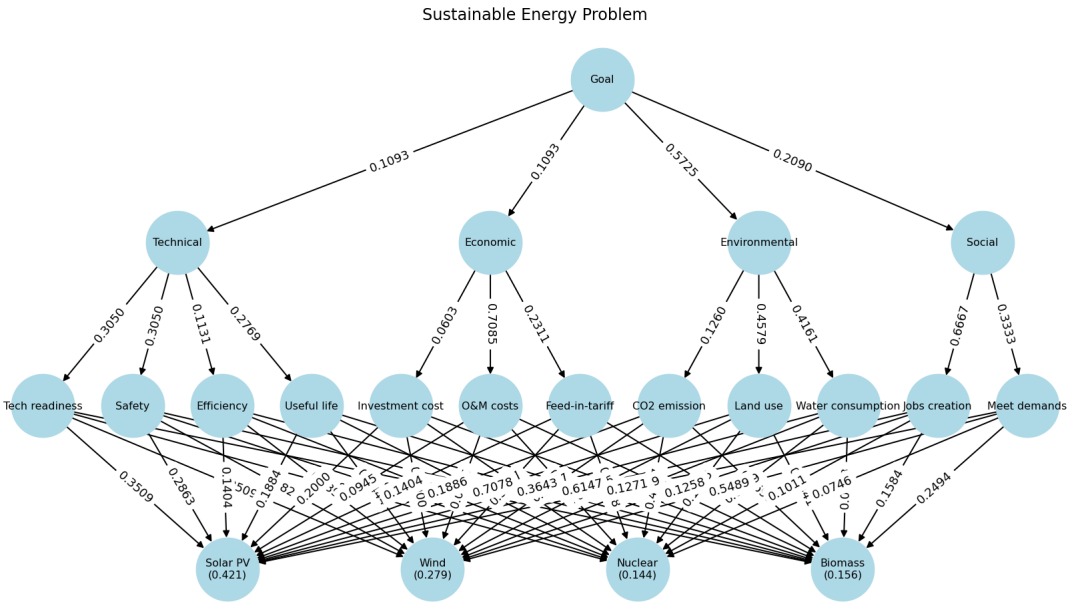
```
Investment cost    0.060328
O&M costs          0.708524
Feed-in-tariff     0.231148
dtype: float64

CI = 0.035633
CR = 0.061436

--------------------------------------------------
Alternatives under Investment cost
          Solar PV  Wind  Nuclear  Biomass
Solar PV       1.0   0.5      1.0      1.0
Wind           2.0   1.0      2.0      2.0
Nuclear        1.0   0.5      1.0      1.0
Biomass        1.0   0.5      1.0      1.0

Alternative Priorities
Solar PV    0.2
Wind        0.4
Nuclear     0.2
Biomass     0.2
dtype: float64

--------------------------------------------------
Alternatives under O&M costs
          Solar PV  Wind   Nuclear   Biomass
Solar PV       1.0   2.0  0.333333  0.142857
Wind           0.5   1.0  0.333333  0.111111
Nuclear        3.0   3.0  1.000000  0.333333
Biomass        7.0   9.0  3.000000  1.000000

Alternative Priorities
Solar PV    0.094454
Wind        0.062475
Nuclear     0.222439
Biomass     0.620632
dtype: float64

--------------------------------------------------
Alternatives under Feed-in-tariff
          Solar PV  Wind  Nuclear  Biomass
Solar PV       1.0   0.5      0.5      0.5
Wind           2.0   1.0      1.0      2.0
Nuclear        2.0   1.0      1.0      1.0
Biomass        2.0   0.5      1.0      1.0

Alternative Priorities
Solar PV    0.140424
Wind        0.339710
Nuclear     0.280848
Biomass     0.239018
```

```
dtype: float64

================================================
Node: Environmental
Pairwise Comparison Matrix
                   CO2 emission  Land use  Water consumption
CO2 emission                1.0      0.25           0.333333
Land use                    4.0      1.00           1.000000
Water consumption           3.0      1.00           1.000000

Local Weights
CO2 emission         0.126005
Land use             0.457934
Water consumption    0.416061
dtype: float64

CI = 0.004601
CR = 0.007933

------------------------------------------------
Alternatives under CO2 emission
          Solar PV  Wind  Nuclear  Biomass
Solar PV  1.000000   0.5      0.5      3.0
Wind      2.000000   1.0      0.5      5.0
Nuclear   2.000000   2.0      1.0      5.0
Biomass   0.333333   0.2      0.2      1.0

Alternative Priorities
Solar PV    0.188624
Wind        0.306684
Nuclear     0.435730
Biomass     0.068963
dtype: float64

------------------------------------------------
Alternatives under Land use
          Solar PV  Wind  Nuclear  Biomass
Solar PV  1.000000  8.00     7.00      9.0
Wind      0.125000  1.00     1.00      4.0
Nuclear   0.142857  1.00     1.00      4.0
Biomass   0.111111  0.25     0.25      1.0

Alternative Priorities
Solar PV    0.707809
Wind        0.122522
Nuclear     0.125540
Biomass     0.044129
dtype: float64

------------------------------------------------
Alternatives under Water consumption
```

```
          Solar PV      Wind  Nuclear  Biomass
Solar PV  1.000000  0.500000      9.0      8.0
Wind      2.000000  1.000000      9.0      9.0
Nuclear   0.111111  0.111111      1.0      0.5
Biomass   0.125000  0.111111      2.0      1.0

Alternative Priorities
Solar PV    0.364338
Wind        0.532925
Nuclear     0.041935
Biomass     0.060802
dtype: float64

================================================
Node: Social
Pairwise Comparison Matrix
               Jobs creation  Meet demands
Jobs creation            1.0           2.0
Meet demands             0.5           1.0

Local Weights
Jobs creation    0.666667
Meet demands     0.333333
dtype: float64

CI = 0.000000
CR = 0.000000

------------------------------------------------
Alternatives under Jobs creation
          Solar PV  Wind  Nuclear  Biomass
Solar PV  1.000000   5.0      6.0      4.0
Wind      0.200000   1.0      1.0      1.0
Nuclear   0.166667   1.0      1.0      0.5
Biomass   0.250000   1.0      2.0      1.0

Alternative Priorities
Solar PV    0.614710
Wind        0.125778
Nuclear     0.101106
Biomass     0.158406
dtype: float64

------------------------------------------------
Alternatives under Meet demands
          Solar PV      Wind  Nuclear   Biomass
Solar PV       1.0  0.200000      2.0  0.500000
Wind           5.0  1.000000      7.0  2.000000
Nuclear        0.5  0.142857      1.0  0.333333
Biomass        2.0  0.500000      3.0  1.000000
```

```
Alternative Priorities
Solar PV    0.127130
Wind        0.548921
Nuclear     0.074592
Biomass     0.249357
dtype: float64

================================================
Alternatives' Global Weights
================================================
Solar PV    0.421290
Wind        0.279321
Nuclear     0.143571
Biomass     0.155818
dtype: float64
```

```
model.plot_hierarchy(title='Sustainable Energy System Problem',
   figsize=(16,8))
```



11. AHP Rating Method

11.1 Employee Evaluation Problem

```
# Employee rating example with data from Excel (2026 04 07)
from DApy.mcdm import AHPRating
import numpy as np
import pandas as pd

criteria = ['Quality','Education','Experience','Dependability']

criteria_pcm = np.array([
    [1,   5,   7,  3],
    [1/5, 1,   3, 1/3],
    [1/7, 1/3, 1,  1/5],
    [1/3,  3,  5,   1]])
```

```


rating_scales = {
    'Quality':   ['Excellent', 'Good', 'Meet Requirements', 'Need
       Improvements', 'Unsatisfactory'],
    'Education': ['Postgraduate', 'Bachelor', 'Diploma','Below Dip'],
    'Experience':['>10 years', '5-10 years', '3-5 years', '1-3 years'
       , '<1 year'],
    'Dependability': ['Exceptional', 'Good', 'Satisfactory', 'Poor']
    }

# Set up Rating System
rating_pcms = {
    'Quality': np.array([
        [ 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]
        ]),
    'Education': np.array([
        [ 1,   3,   5,  9],
        [1/3,  1,   3,  7],
        [1/5, 1/3,  1,  3],
        [1/9, 1/7, 1/3, 1]
        ]),
    'Experience': np.array([
```

```
        [ 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]
        ]),
    'Dependability': np.array([
        [ 1,   3,   5,  9],
        [1/3,  1,   3,  5],
        [1/5, 1/3,  1,  3],
        [1/9, 1/5, 1/3, 1]
        ])
    }
```

```
# Set up Rating System
emp_eval = AHPRating(criteria, criteria_pcm, rating_scales,
   rating_pcms)

# See rating system details
emp_eval.rating_system_details()
```

```
=== Criteria Weights ===
    Criterion   Weight
      Quality 0.565009
    Education 0.117504
   Experience 0.055285
Dependability 0.262201

=== Criteria Pairwise Comparison Matrix ===
              Quality Education Experience Dependability
Quality             1         5          7             3
Education         1/5         1          3           1/3
Experience        1/7       1/3          1           1/5
Dependability     1/3         3          5             1
λ_max = 4.116982, CI = 0.038994, CR = 0.043327

=== Rating Details ===

--- Criterion: Quality ---

Rating Weights (Idealized):
           Rating  Ideal Weight
        Excellent      1.000000
             Good      0.787997
Meet Requirements      0.317927
Need Improvements      0.140058
```

```
   Unsatisfactory      0.086395

PCM (Fraction Form):
                  Excellent Good Meet Requirements Need Improvements Unsatisfactory
Excellent                 1    1                 5                 7              9
Good                      1    1                 3                 5              7
Meet Requirements       1/5  1/3                 1                 3              5
Need Improvements       1/7  1/5               1/3                 1              2
Unsatisfactory          1/9  1/7               1/5               1/2              1
λ_max = 5.135596, CI = 0.033899, CR = 0.030267

--- Criterion: Education ---

Rating Weights (Idealized):
      Rating  Ideal Weight
Postgraduate      1.000000
    Bachelor      0.472971
     Diploma      0.192225
   Below Dip      0.078621

PCM (Fraction Form):
             Postgraduate Bachelor Diploma Below Dip
Postgraduate            1        3       5         9
Bachelor              1/3        1       3         7
Diploma               1/5      1/3       1         3
Below Dip             1/9      1/7     1/3         1
λ_max = 4.087630, CI = 0.029210, CR = 0.032456

--- Criterion: Experience ---

Rating Weights (Idealized):
    Rating  Ideal Weight
 >10 years      1.000000
5-10 years      0.774648
 3-5 years      0.391653
 1-3 years      0.252030
   <1 year      0.125353

PCM (Fraction Form):
           >10 years 5-10 years 3-5 years 1-3 years <1 year
>10 years          1          1         3         5       7
5-10 years         1          1         2         3       5
3-5 years        1/3        1/2         1         2       3
1-3 years        1/5        1/3       1/2         1       3
<1 year          1/7        1/5       1/3       1/3       1
λ_max = 5.087228, CI = 0.021807, CR = 0.019471

--- Criterion: Dependability ---

Rating Weights (Idealized):
      Rating  Ideal Weight
```

```
 Exceptional      1.000000
        Good      0.439823
Satisfactory      0.196547
        Poor      0.086010

PCM (Fraction Form):
             Exceptional Good Satisfactory Poor
Exceptional            1    3            5    9
Good                 1/3    1            3    5
Satisfactory         1/5  1/3            1    3
Poor                 1/9  1/5          1/3    1
λ_max = 4.076293, CI = 0.025431, CR = 0.028257
```

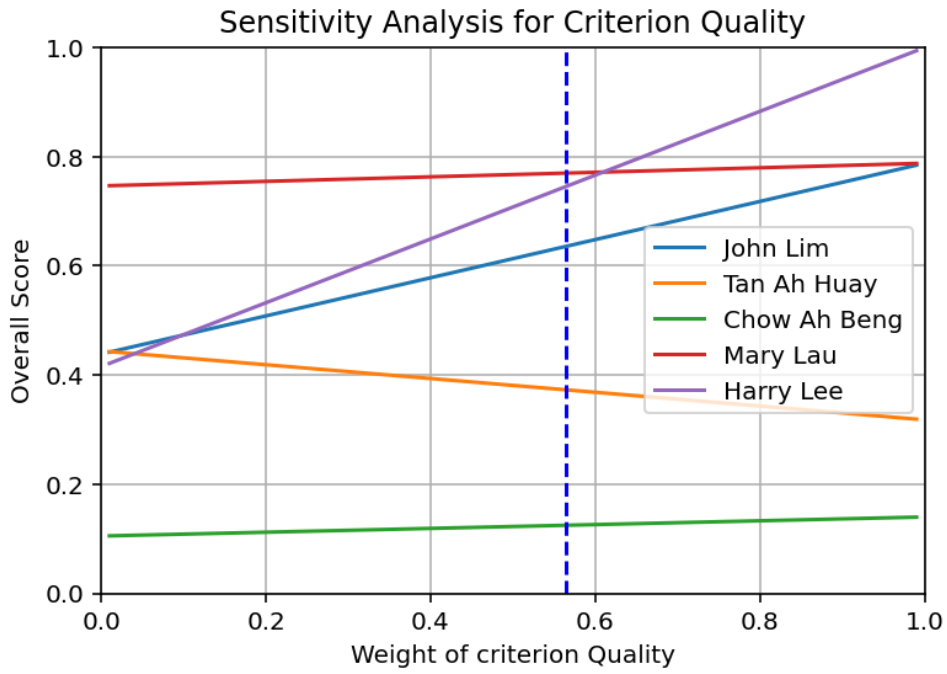
```
# Read candidates rating data from Excel
employee_ratings = AHPRating.read_candidate_ratings(
                     'employee_ratings_data.xlsx', 'employees',
                     display=True)
```

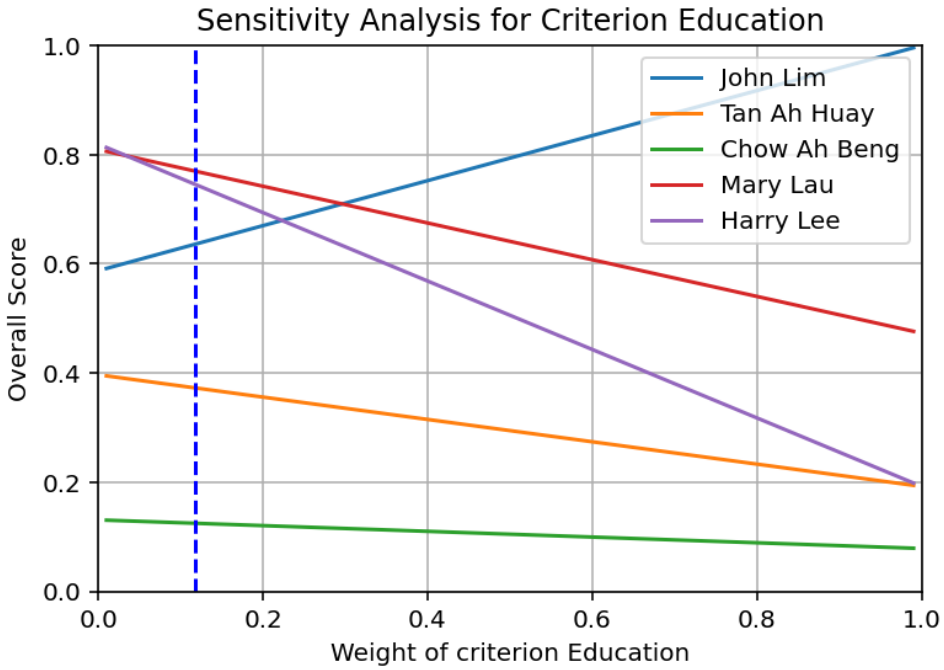
```
=== Candidates Names and Ratings Data ===
John Lim
  Quality       : Good
  Education     : Postgraduate
  Experience    : 3-5 years
  Dependability : Satisfactory
Tan Ah Huay
  Quality       : Meet Requirements
  Education     : Diploma
  Experience    : >10 years
  Dependability : Good
Chow Ah 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
```

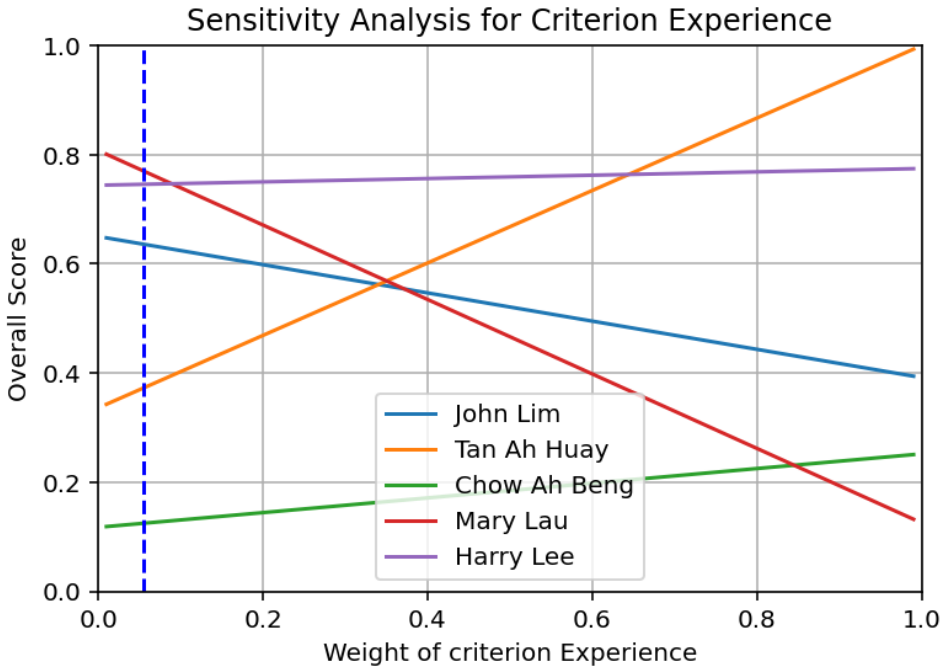
```
eval_results = emp_eval.evaluate(employee_ratings)
# Make sure we see everything in one whole table.
pd.set_option('display.max_columns', 100,
                            'display.max_colwidth', 100,
                            'display.width', 500)
print(eval_results)
```

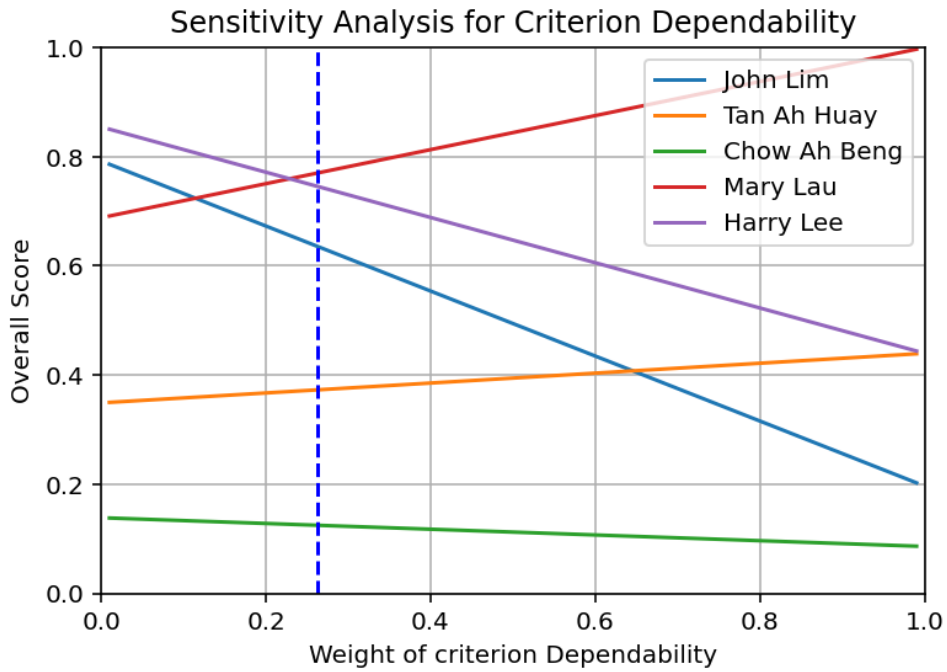
	Name	Overall	Quality	Education	Experience	Dependability
0	Mary Lau	0.769933	0.787997	0.472971	0.125353	1.000000
1	Harry Lee	0.745745	1.000000	0.192225	0.774648	0.439823
2	John Lim	0.635918	0.787997	1.000000	0.391653	0.196547
3	Tan Ah Huay	0.372826	0.317927	0.192225	1.000000	0.439823
4	Chow Ah Beng	0.124858	0.140058	0.078621	0.252030	0.086010

```
# Run sensitivity_analysis on all candidates
emp_eval.sensit()
```



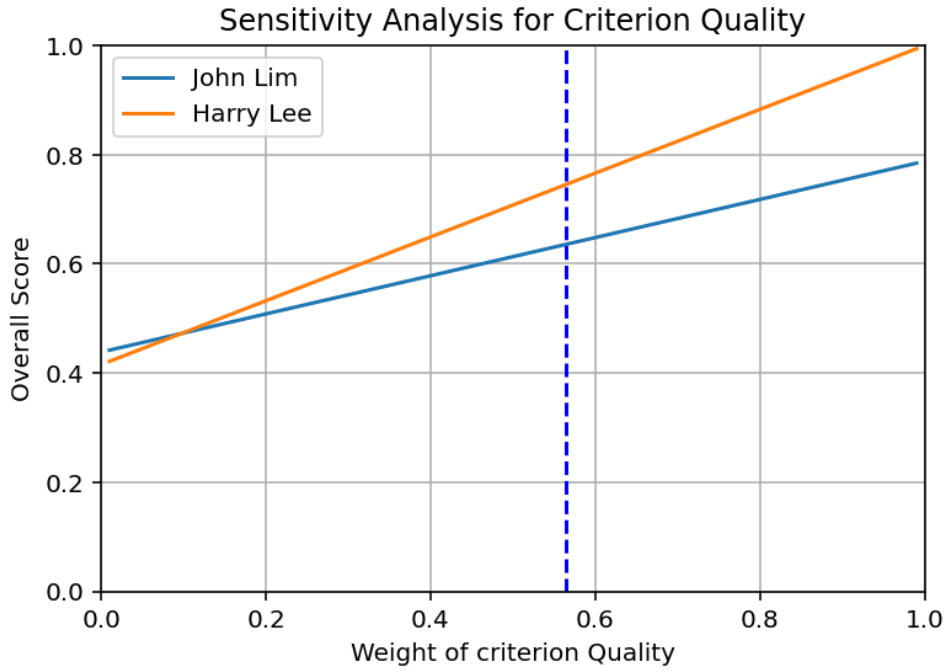

Sensitivity Analysis for Criterion Education
Overall Score
Weight of criterion Education
John Lim
Tan Ah Huay
Chow Ah Beng
Mary Lau
Harry Lee
0.0
0.2
0.4
0.6
0.8
1.0

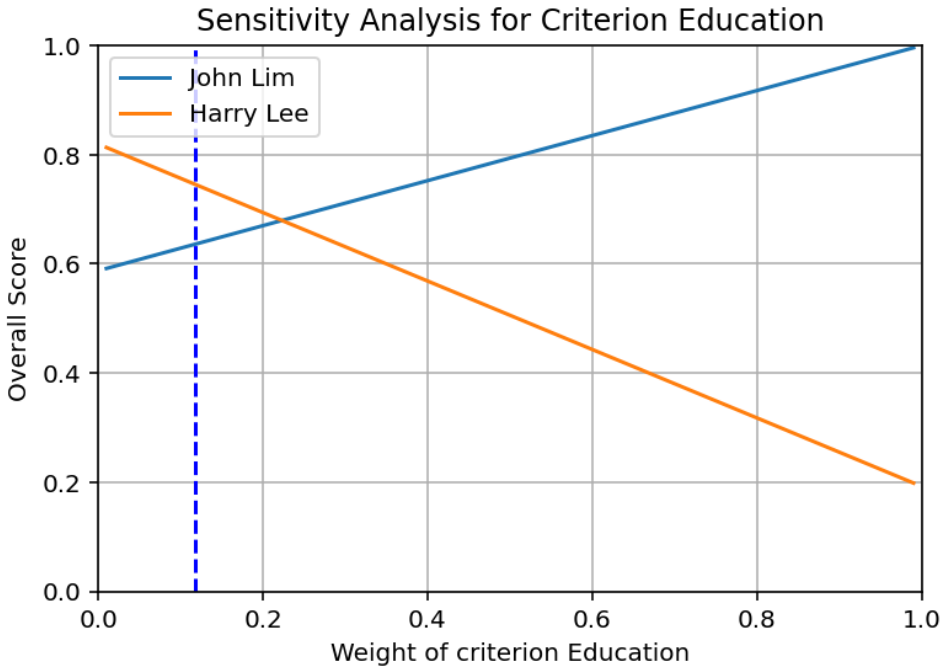

Sensitivity Analysis for Criterion Experience
Overall Score
Weight of criterion Experience
John Lim
Tan Ah Huay
Chow Ah Beng
Mary Lau
Harry Lee
0.0
0.2
0.4
0.6
0.8
1.0

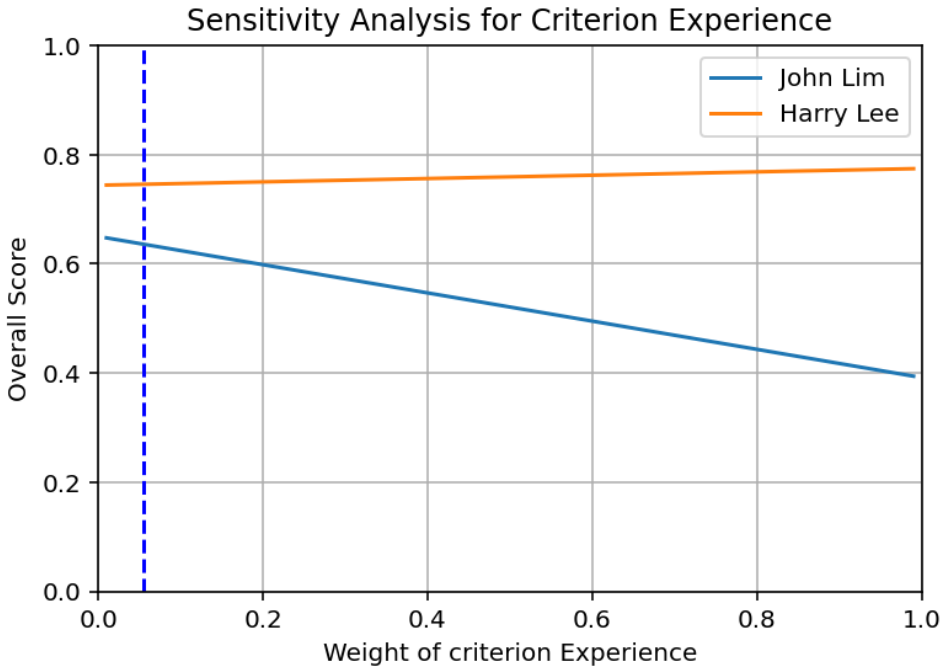


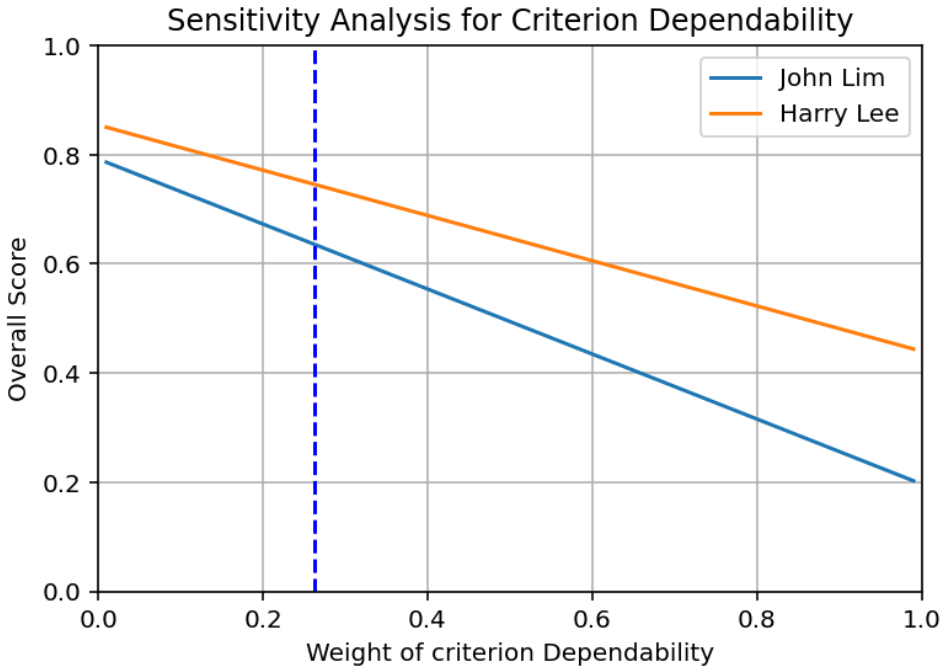
Compare only John Lim and Harry Lee in the sensivity analysis.

```
# Run sensitivity on specific list of candidates
emp_eval.sensit(['John Lim','Harry Lee'])
```




Sensitivity Analysis for Criterion Education
John Lim
Harry Lee
Overall Score
Weight of criterion Education
0.0
0.2
0.4
0.6
0.8
1.0


Sensitivity Analysis for Criterion Experience
John Lim
Harry Lee
Overall Score
Weight of criterion Experience
0.0
0.2
0.4
0.6
0.8
1.0


Sensitivity Analysis for Criterion Dependability
John Lim
Harry Lee
Overall Score
Weight of criterion Dependability
0.0
0.2
0.4
0.6
0.8
1.0

12. Cost-Effectiveness Analysis

12.1 The Drug Counselling Center Relocation Problem

We first analyse the effectiveness of the alternative relation sites using AHP without consideration of costs.

```
from DApy.mcdm import AHPNode, AHPModel
import numpy as np

goal_pcm = np.array([
    [ 1,   2,  2, 3],
    [1/2,  1,  1, 2],
    [1/2,  1,  1, 1],
    [1/3, 1/2, 1, 1]
])

goal = AHPNode("Goal", pcm=goal_pcm)

C1 = AHPNode('Good conditions for staff')
C2 = AHPNode('Easy access for clients')
C3 = AHPNode('Suitability of space')
C4 = AHPNode('Administrative convenience')

goal.children = [C1,C2,C3,C4]

# Subcriteria for C1 'Good conditions for staff'
C11 = AHPNode('Office size')
C12 = AHPNode('Staff commuting')
C13 = AHPNode('Office attractiveness')
C14 = AHPNode('Office privacy')
C15 = AHPNode('Staff parking')
C1.children = [C11,C12,C13,C14,C15]
C1.pcm = np.array([
    [ 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]
 ])

# Subcriteria for C2 'Easy access for clients'
C21 = AHPNode("Close to client's home")
C22 = AHPNode('Access to public transport')
```

```
C2.children = [C21,C22]
C2.pcm = np.array([
    [1, 1],
    [1, 1]
])

# Subcriteria for C3 'Suitability of space'
C31 = AHPNode('Counceling rooms')
C32 = AHPNode('Conference rooms')
C33 = AHPNode('Reception area')
C3.children = [C31,C32,C33]
C3.pcm = np.array([
  [ 1,   2,  2],
  [1/2,  1,      2],
  [1/2, 1/2, 1]
])

# Subcriteria for C4 'Administrative convenience'
C41 = AHPNode('Adequacy of space')
C42 = AHPNode('Flexibilty of space')

C4.children = [C41,C42]
C4.pcm = np.array([
    [ 1,  2],
    [1/2, 1]
 ])
```

The altenative sites and their pairwise comparsion matrices.

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

# Alternative PCMs
C11.alt_pcm = np.array([
    [ 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 ]
])

C12.alt_pcm = np.array([
    [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 ]
])

C13.alt_pcm = np.array([
    [ 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 ]
])

C14.alt_pcm = np.array([
    [ 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 ]
])

C15.alt_pcm = np.array([
    [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 ]
])

C21.alt_pcm = np.array([
    [ 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 ]
])

C22.alt_pcm = np.array([
    [ 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 ]
])

C31.alt_pcm = np.array([
    [ 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 ],
])

C32.alt_pcm = np.array([
    [ 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 ],
])

C33.alt_pcm = np.array([
    [ 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 ]
])

C41.alt_pcm = np.array([
    [ 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 ],
])

C42.alt_pcm = np.array([
    [ 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 ]
])
```

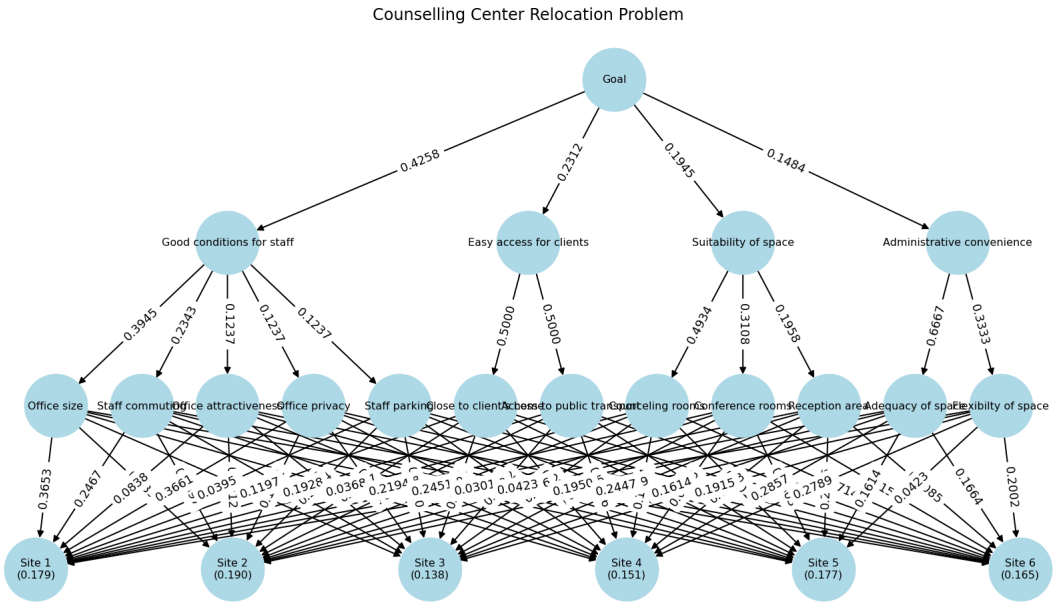
Build the AHP model and solve it for the effectiveness scores.

```
eff_model = AHPModel(goal, alternatives)
eff_scores = eff_model.solve()
```

```
Alternative     Global weight
  Site 1          0.179105
  Site 2          0.190379
  Site 3          0.137681
  Site 4          0.150873
  Site 5          0.176830
  Site 6          0.165132
```

Plot the AHP effectiveness model.

```
eff_model.plot_hierarchy(
        title='Counselling Center Relocation Problem',
        figsize=(16,8))
```



Perform Cost-Effectivess Analysis using class `ParetoAnalysis`.

```
from DApy.mcdm import ParetoAnalysis

# The cost data
costs = {
 'Site 1': 48.0,
 'Site 2': 53.3,
 'Site 3': 54.6,
 'Site 4': 60.6,
 'Site 5': 67.8,
 'Site 6': 47.5 }

# Built {key: (cost, effectiveness)} dict
cost_eff_data = {k: (costs.get(k), eff_scores.get(k))
                     for k in eff_scores.keys() | costs.keys()}

labels = ['EUAC ($K)', 'Effectiveness']
```

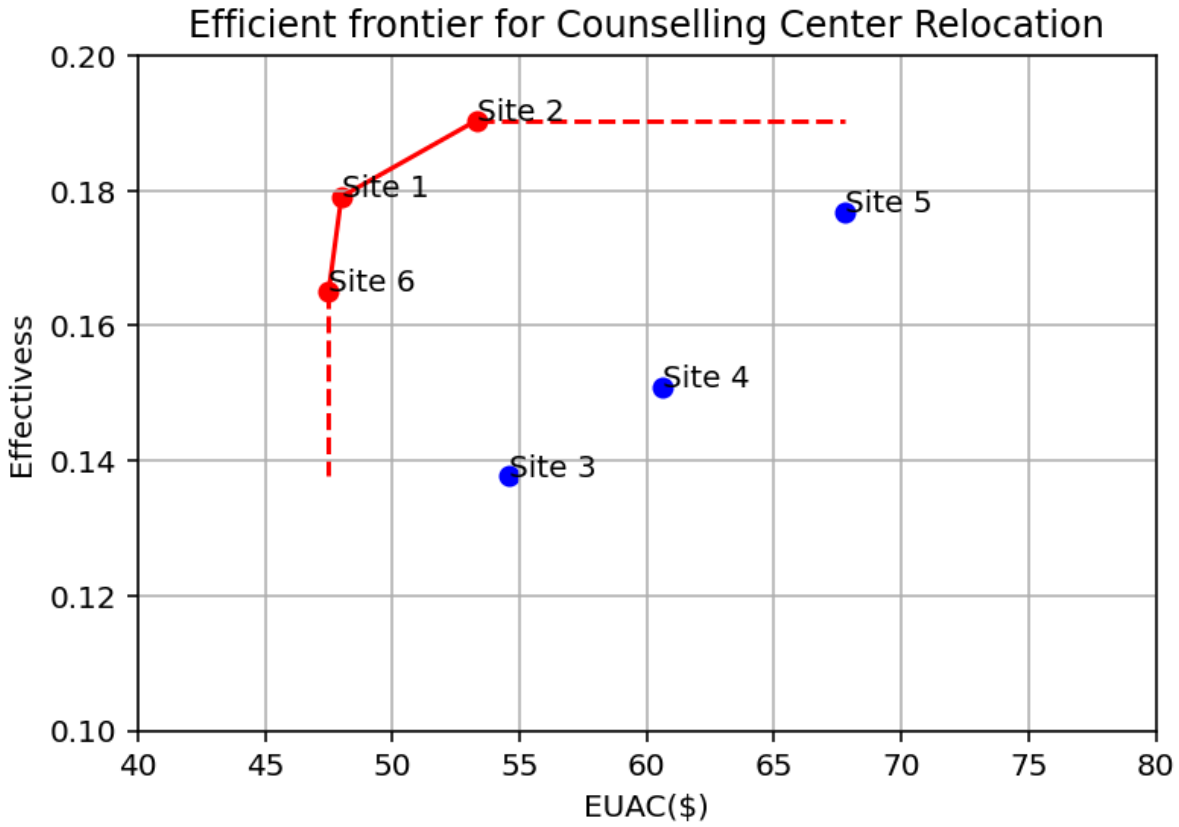
Peform the Pareto efficiency analysis.

```
cost_eff_model = ParetoAnalysis(cost_eff_data, ['min','max'],
                    labels=['EUAC($)','Effectivess'])

eff_sites = cost_eff_model.efficient_points()

# Sort eff solutions by increasing cost
sorted_sites = dict(sorted(eff_sites.items(),
                    key=lambda item: item[1][0]))
for k, v in sorted_sites.items():
    print(f"{k:3}: {', '.join([f'{x:6.4f}' for x in v])}")
```

```
Site 6: 47.5000, 0.1651
Site 1: 48.0000, 0.1791
Site 2: 53.3000, 0.1904
```

Plot the efficient frontier.

```
ax = cost_eff_model.plot_frontier()
# Customise the axes any way you like
ax.set_title("Efficient frontier for Counselling Center Relocation")
ax.set_xlim(40,80)
ax.set_ylim(0.1,0.2)
ax.grid()
plt.show()
```



13. Read the Docs

13.1 Class: risk.DiscreteDeal

The `risk.DiscreteDeal` class supports the analysis of discrete deals. In addition to providing methods for computing the expected value, standard deviation, and the variance of the deal, it plots risk profiles in PMF, CDF, and EPF format. First and second-order stochastic dominance analysis between two deals can be carried out.

```
class risk.DiscreteDeal(x, p, states=None, name=None)

  Parameters:
    x : array of x values
    p: array of probabilities
    states:  an optional list of outcome states
    name:  an optional name of the deal
  Returns:  class object

  Methods:
    ev()
     Returns:  the expected value of the deal
    var()
      Returns: the variance of the deal
    std()
      Returns:  the standard deviation of the deal

    plot_pmf(xlim=None, ax=None)
      Plot the risk profile as a PMF
      Parameters:
        xlim:  optional plot limits tuple (L, H)
        ax:  optional Axes object to be modified
      Returns:  an Axes object

    plot_cdf(xlim=None, ax=None)
      Plot the risk profile as a CDF.
      Parameters:
        xlim: optional plot limits tuple (L, H)
        ax:  optional Axes object to be modified
      Returns:  an Axes object

    plot_epf(xlim=None, ax=None)
      Plot the risk profile as an EPF.
      Parameters:
        xlim:  optional plot limits tuple (L, H)
        ax:  optional Axes object to be modified.
      Returns:  an Axes object
```

```
    first_order_SD(cls, A, B, xlim=None, plot_cdf=True, plot_epf=True
       )
      Determine if A first-order stochastically dominates B
      Parameters:
         A:  a DiscreteDeal object
         B:  a DiscreteDeal object
         xlim: optional tuple indicating the range of x values to
            check and plot
         plot_cdf: optional boolean to indicate plotting the the CDF
                 plot_epf: optional boolean to indicate plotting the
                    the EPF
       Returns: Boolean

    second_order_SD(A, B, xlim=None, plot=True)
       Determine if A second-order stochastically dominates B
       Parameters:
         A:  a DiscreteDeal object
         B:  a DiscreteDeal object
         xlim: optional tuple indicating the range of x values to
            check and plot
         plot: optional boolean to indicate plotting the risk
            profiles and relevant charts
       Returns: Boolean

    PISP(w0, wealth_utility_fn, x0=None, x1=None, bracket=None)
      Compute the personal indifferent selling price or certainty
         equivalent
      Parameters:
        w0: initial wealth,
        wealth_utility_fn:  callable wealth utility function
        x0: optional initial guess
        bracket: bounds with a sign change in the function value
      Returns: personal indifferent selling price or CE of the deal

    PIBP(w0, wealth_utility_fn, x0=None, bracket=None)
      Compute the personal indifferent buying price
      Parameters:
        w0: initial wealth,
        wealth_utility_fn:  callable wealth utility function
        x0: optional initial guess
        bracket: bounds with a sign change in the function value
      Returns: personal indifferent buying price of the deal
```

13.2 Class: bn.DSepAnalysis

The `bn.DSepAnalysis` class performs d-separation analysis on directed acyclic graphs (DAG).

```
class bn.DSepAnalysis(G)
  Parameters:
    G:  a networkx directed graph
  Returns:  class object

   Methods:
     plot(X=None, Y=None, Z=None,  pos=None, title=None, figsize=None
        )
       Parameters:
         X:  optional list of nodes in set X
         Y:  optional list of nodes in set Y
         Z:  optional list of nodes in set Z
         pos: optional dict of node positions to plot.  Default
            spring layout.
         title: optional title for the plot
         figsize: optional figure size (tuple) of the plot
      Returns: None

      is_d_separated(X, Y, Z)
        Parameters:
           X:  list of nodes in set X
           Y:  list of nodes in set Y
           Z:  list of nodes in set Z
        Returns:  Boolean

      path_blocking_analysisX, Y, Z, cutoff=20, max_paths=50)
        Parameters:
           X:  list of nodes in set X
           Y:  list of nodes in set Y
           Z:  list of nodes in set Z
        Returns:  A path-by-path analysis of all paths from X to Y,
           blocking by Z
```

13.3 Class: cara.ExponUtilityFunction)

The `cara.ExponUtilityFunction` class supports the ploting of exponential utility functions and assessment of risk tolerance using the 50:50 CE method.

```
class cara.ExponUtilityFunction(RT=None)
  The exponential utility function under constant absolute risk
     aversion (CARA).
  Parameters:
    RT: optional risk tolerance for the utility function.  Must be
       provided later.
    Returns:  object instance

  Methods:
    plot(L, H, ax=None, label=None, num=200)
      Parameters:
        L: lower bound of function
        H: upper bound of function
        ax: optional Axes object
        label: optional label
        num:  optional number of points of plot.
    Returns:  an Axes object.

    get_function(L, H)
      Parameters:
        L, H: boundary conditions such that u(L)=0, u(H)=1.
      Returns:  a callable function.

    RT_from_CE50(L, H, CE50, x0=None, bracket=None)
      Parameters:
        L, H: boundary conditions such that u(L)=0, u(H)=1.
       CE50: the certainty equivalent with utility = 0.5
       x0:  optional initial guess of RT
       bracket: optional bracket for RT.
         Either X0 or the bracket must be provided.
      Returns:  The risk tolerance

    CE(u_val, L, H)
      Compute the certainty equivalent
      Parameters:
        u_val: the utility whose CE is computed
        L, H:  boundary condtions such that u(L)=0, u(H)=1
      Returns: the certainty equivalent
```

13.4 Class: probability.FitPDF)

The `probability.FitPDF` class supports the fitting of continuous probability distributions to data. It directly uses the scipy.stats fitting methods.

```
class probability.FitPDF(data)
  Fit continuous distributions to data
  Parameters:
    data: a 1-D array of data
  Returns:  object instance

  Methods:
    fit( candidate_distributions=None, method='MLE')
      Parameters:
        candidate_distributions:  optional list of valid scipy.stats
           probability distribution names
        method= optional method of fitting. Default MLE. Other method
           : MM
     Returns:  dataframe of fit results

      results(sort='KS', top_k=20)
        Parameters:
           sort: optional fit metric to sort, default KS
           top_k: optional number of top results to show, default 20.
      Returns:  Fit results

      plot_pdf(sort='KS', top_k=10):
        Parameters:
           sort: optional fit metric to sort, default KS
           top_k: optional number of top results to show, default 10.
        Returns:  None

        plot_cdf(sort='KS', top_k=10):
        Parameters:
           sort: optional fit metric to sort, default KS
           top_k: optional number of top results to show, default 10.
        Returns:  None

      save_results_to_excel(file_name='outputs.xlsx', sort='KS'):
        Parameters:
           file_name:  optional Excel file name.  Default outputs.
              xlsx
           sort: optional fit metric to sort, default KS
        Returns:  None
```

13.5 Class: probability.FitPMF

The `probability.FitPMF` class supports the fitting of discrete probability distributions to data. Since scipy.stats does not support direct fitting of discrete probability distribution, we do our own MLE optimization here. The challenge is in finding good initial good estimates or brackets for the parameters.

```
class probability.FitPMF(data)
  Fit continuous distributions to data
  Parameters:
    data: a 1-D array of data
  Returns:  object instance

  Methods:
    fit(candidate_list=None)
    Parameters:
      candidate_list: optional list of candidate distributions fit.
         The default list has 13.
    Returns:  dataframe of fit results

    results(sort='KS', top_k=10)
        Parameters:
           sort: optional fit metric to sort, default KS
           top_k: optional number of top results to show, default 20.
      Returns:  Fit results

      plot_pmf(sort='KS', top_k=10):
        Parameters:
           sort: optional fit metric to sort, default KS
           top_k: optional number of top results to show, default 10.
        Returns:  None

        plot_cdf(sort='KS', top_k=10):
        Parameters:
           sort: optional fit metric to sort, default KS
           top_k: optional number of top results to show, default 10.
        Returns:  None

      save_results_to_excel(file_name='outputs.xlsx', sort='KS'):
        Parameters:
           file_name:  optional Excel file name.  Default outputs.
              xlsx
           sort: optional fit metric to sort, default KS
        Returns:  None
```

13.6 Class: probability.Fit_norm_percentiles

The `probability.Fit_norm_percentiles` class fits normal distribution to percentile data.

```
class Fit_norm_percentiles(xs, ps)
  Fit Normal Distributions to percentile data.
   Parameters:
     xs:  array-like x values
     ps:  array-like percentiles p (0 < p < 1)

  Methods:
    fit()
       Parameters: None
       Returns: mu, sigma, r_squared (for n>2)

    plot()
      Plot the results
        Parameters: None
        Return: Axes object
```

13.7 Class: mcdm.AHPMatrix

The `mcdm.AHPMatrix` class supports the computation of pairwise comparison matrices in AHP. Computational methods included are 'eigen', 'algebra', 'power', 'rgm', and 'column'. The later two are approximatation methods. They are included here for comparison of results with other exact or more precise methods.

```
class mcdm.AHPMatrix(pcm, labels=None)
   Compute AHP matrix using different methods
   Parameters:
     pcm: A valid AHP matrix
     labels: optional list of labels for the rows/columns
  Return:  Object instance

    Methods:
    compute(method='eigen')
       Compute the AHP matrix
       Parameters:
         method = 'eigen' (default), 'rgm', 'column', 'power', '
            algebra'
      Returns:  w, lambda_max, CI, CR

    matrix()
      Pretty print the matrix
        Parameters: None
        Returns:  the matrix

      preference_graph()
        Draw the preference graph and detect directed cycles.
      Parameters: None
      Returns: None
```

13.8 Class: mcdm.AHPNode

The `mcdm.AHPNode` class is used to define and create the criteria nodes in an AHP model.

```
class mcdm.AHPMatrix(name, children)
    Define an AHP criterion node
    Parameters:
      name:  name of the node
      children: list of AHPNode objects
    Returns:  object instance
```

13.9 Class: mcdm.AHPModel

The `mcdm.AHPModel` class supports the modeling and analysis of AHP models with tree hierarchy. The criteria may have sub-criteria, and the tree may be inbalance depth.

```
class AHPModel(goal_node, alternatives)
  Define a general AHP model
  Parameters:
    goal_node:  an AHPNode object.
    alternatives:  a list of the alternatives
  Returns:  object instance

  Methods:
    solve()
      Solve the AHP model
      Parameters:  None
      Returns: the global weights

    plot_global_weights()
      Plot the global weight of alternatives as a histogram
      Parameters:  None
      Returns: None

    display_model()
       Display all the matrices, local weights, CI \& CR values
       Parameters:  None
       Returns: None

    plot_text_tree()
       Display the AHP tree in text format.
                Parameters:  None
       Returns: None

    sensit()
        Perform sensitivity analysis and plot rainbow diagrams
        Parameters:  None
        Returns: None

     plot_hierarchy(title="AHP Hierarchy", figsize=(10,6))
        Plot the AHP hierarchy using networkx
        Parameters:
          title: optional hierarchy title.  default: AHP Hierarchy
          figsize: optional figure size, default(10,6)
        Returns: None
```

13.10 Class: mcdm.AHPRating

The `mcdm.AHPRating` class supports the rating method in AHP. It can be used for the assessment or ranking of a large number of candidates (alternatives).

```
class mcdm.AHPRating(criteria, criteria_pcm, rating_scales,
   rating_pcms)
   Perform AHP Rating Method for evaluation of candidates
   Parameters:
      criteria: list of criteria
      criteria_pcm: the criteria pairwise comparison matrices
      rating_scales: the rating scales
      rating_pcms: the rating pairwise comparison matrices
   Returns:  Object instance

   Methods:
     rating_system_details(display=True, labeled_pcm=True)
        Print Rating system details
        Parameters:
           display: optional flag
           labeled_pcm: optional flag
        Returns: criteria dataframe, ratings dataframe, pcm
           information

    evaluate(candidate_ratings)
      Evaluate the list of candidates
      Parameters:
         candidate_ratings:  the candidate ratings
      Returns:  results in a dataframe

     sensit(candidate_names=None, steps=50):
        Perform Sensitivity Analysis
        Parameters:
          candidate_names:  list of candidates to include the
             sensitivity analysis
        Returns: None

        read_candidate_ratings(file_name, sheet_name, display=False)
          Read the candidate ratings from an Excel file
          Parameters:
            file_name: the Excel file name
            sheet_name:  the Excel sheet name
            display: optional flag
          Returns:  candidate ratings that can be read by evaluate().
```

13.11 Class: mcdm.ParetoAnalysis

The `mcdm.ParetoAnalysis` perform pareto optimality or efficiency analysis up to n attributes. Directions of preference for each attribute may be specified as either "min" or "max". No prior conversion of all to "max" direction is required. Efficient frontier plots are provided when $n = 2$ or $n = 3$.

```
class mcdm.ParetoAnalysis(data, directions, labels=None)
  Perform Parento Optimal Analysis
  Parameters:
    data: {name: (x1, x2, ..., xn)}
    directions: ("max"/"min", ...)
    labels: [name1, name2, ...] optional default ['x1','x2',...]

    Methods:
      efficient_points()
        Find the set of efficient points
        Parameters:  None
        Returns: dictionary

      is_efficient(x)
         Determine if x is an efficient point
         Parameters:
            x: the point to be evaluated
         Returns: Boolean

     plot_frontier()
         Plot the efficient frontier
         Parameters: None
         Returns: None
```

13.12 Function: utils.read_data_from_excel

The `utils.read_data_from_excel()` function reads data from a named excel file and returns a flatten numpy array.

```
function: utils.read_data_from_excel(file_name, shee_name)
  Parameters:
    file_name: an excel file
    sheet_name: an excel worksheet.
  returns:
    A numpy array.
```