EMA_Workbench_Waste_Management_China_Final

January 31, 2017

In [ ]:
```
from ema_workbench import ema_logging
```

In [ ]:
```
#Import packages, functions

from __future__ import unicode_literals, absolute_import
from ema_workbench.connectors import pyNetLogo
import matplotlib.pyplot as plt


from ema_workbench.connectors.netlogo import NetLogoModel

from ema_workbench.em_framework import (TimeSeriesOutcome, RealParameter,
                                        perform_experiments, IntegerParamet
#from ema_workbench.em_framework.parameters import Parameter
from ema_workbench.util import ema_logging
from ema_workbench.analysis import plotting, plotting_util
from ema_workbench.analysis.pairs_plotting import pairs_lines, pairs_scatte
from ema_workbench.util.utilities import save_results, load_results
from ema_workbench.analysis.prim import setup_prim

import mpld3

import ema_workbench.analysis
import sys
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

from ema_workbench.analysis.plotting import lines, envelopes
from ema_workbench.analysis.plotting_util import VIOLIN, KDE
```

In [ ]:
```
#Set-up of the experiment
ema_logging.log_to_stderr(ema_logging.INFO)

#Connect to the netlogo model
model = NetLogoModel('WasteMGMT',
                     wd = "./models/Waste MGMT",
```

```
                                model_file="Recycling Behavior - Version 1.6 Pyth

        #Set number ticks
        model.run_length = 1000

        #define uncertainties
        model.uncertainties = [RealParameter("WeightOfAttitude", 1, 5 ),
                               RealParameter("WeightOfSocialNorm", 1, 5),
                               RealParameter("WeightOfPerceivedBehaviouralControl"
                              RealParameter("ContainerIncentives",1,30),
                              RealParameter("CollectorIncentives",1,30),
                              RealParameter("TransformationIntoBehaviour",50,100),
                              IntegerParameter("TheoryOfBehaviour",1,4), #Hack to
                              IntegerParameter("seed",1,9999) #Including Problisti
                              ]

        #Define model outcomes
        model.outcomes = [TimeSeriesOutcome('ParticipationRateLandfill'),
                          TimeSeriesOutcome('ParticipationRateContainer'),
                          TimeSeriesOutcome('ParticipationRateCollector')]


        #perform experiments
        n = 2000

        #My laptop cannot run the models in parallel. So it is set to false.
        results = perform_experiments(model, n, parallel = False, reporting_interva
        ema_logging.info('runs completed')
        save_results(results, r'ChinaWasteMGMTv2 {}.tar.gz'.format(n))

In [ ]: #load the results
        results = load_results("ChinaWasteMGMTv2 2000.tar.gz")
        experiments, outcomes = results

        #Plot for exploration
        data = outcomes['ParticipationRateLandfill']

        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(np.mean(data, axis=0), label = 'mean')
        ax.fill_between(np.arange(0, data.shape[1]),np.min(data, axis=0), np.max(da
        ax.plot(np.percentile(data, 25, axis=0), label='first quartile')
        ax.plot(np.percentile(data, 75, axis=0), label='third quartile')
        plt.title("Participation Rate Landfill")
        plt.ylabel("Participation Rate")
        plt.xlabel("Days")

        ax.legend()
```

```
        plt.show()

In [ ]: #plot for exploration
        data = outcomes['ParticipationRateContainer']

        print(data)
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(np.mean(data, axis=0), label = 'mean')
        ax.fill_between(np.arange(0, data.shape[1]),np.min(data, axis=0), np.max(da
        ax.plot(np.percentile(data, 25, axis=0), label='first quartile')
        ax.plot(np.percentile(data, 75, axis=0), label='third quartile')

        ax.legend()
        plt.show()

In [ ]: #plot for exploration
        data = outcomes['ParticipationRateCollector']

        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(np.mean(data, axis=0), label = 'mean')
        ax.fill_between(np.arange(0, data.shape[1]),np.min(data, axis=0), np.max(da
        ax.plot(np.percentile(data, 25, axis=0), label='first quartile')
        ax.plot(np.percentile(data, 75, axis=0), label='third quartile')

        ax.legend()
        plt.show()

In [ ]: #furter exploration of the data. Plotting timeseries and outcomes density a
        experiments, outcomes = results
        oois = outcomes.keys()

        for ooi in oois:
            data_to_sort_by = outcomes[ooi][:,-1]
            indices = np.argsort(data_to_sort_by)
            indices = indices[1:indices.shape[0]:20] #last number is used to determ

            lines(results, outcomes_to_show=ooi, density=VIOLIN,
                  show_envelope=True, experiments_to_show=indices)

        plt.show()

In [ ]: #Setting up variables for a correlation test. Is used for the validation of
        PLandfill = outcomes['ParticipationRateLandfill'][:,-1]
        PCollector = outcomes['ParticipationRateCollector'][:,-1]
        PContainer = outcomes["ParticipationRateContainer"][:,-1]
```

```
CollectorIncentives = experiments['CollectorIncentives']
ContainerIncentives = experiments['ContainerIncentives']
WAttitude = experiments["WeightOfAttitude"]
WSocialNorm = experiments['WeightOfSocialNorm']
WBehaviouralControl = experiments['WeightOfPerceivedBehaviouralControl']
```

In [ ]:
```
#Plotting scatterplots with correlation test. Is used for validation of the
a = sns.jointplot(PLandfill,CollectorIncentives, size=5)
a.set_axis_labels("ParticipationRateLandfill", "CollectorIncentives")

b = sns.jointplot(PLandfill,ContainerIncentives, size=5)
b.set_axis_labels("ParticipationRateLandfill", "ContainerIncentives")

c = sns.jointplot(PLandfill,WAttitude, size=5)
c.set_axis_labels("ParticipationRateLandfill", "WeightOfAttitude")

d = sns.jointplot(PLandfill,WSocialNorm, size=5)
d.set_axis_labels("ParticipationRateLandfill", "WeightOfSocialNorm")

e = sns.jointplot(PLandfill,WBehaviouralControl, size=5)
e.set_axis_labels("ParticipationRateLandfill", "WeightOfPerceivedBehavioura

## Graphs correlation with Participation Rate collector

f = sns.jointplot(PCollector,CollectorIncentives, size=5)
f.set_axis_labels("ParticipationRateCollector", "CollectorIncentives")

g = sns.jointplot(PCollector,ContainerIncentives, size=5)
g.set_axis_labels("ParticipationRateCollector", "ContainerIncentives")

h = sns.jointplot(PCollector,WAttitude, size=5)
h.set_axis_labels("ParticipationRateCollector", "WeightOfAttitude")

j = sns.jointplot(PCollector,WSocialNorm, size=5)
j.set_axis_labels("ParticipationRateCollector", "WeightOfSocialNorm")

k = sns.jointplot(PCollector,WBehaviouralControl, size=5)
k.set_axis_labels("ParticipationRateCollector", "WeightOfPerceivedBehaviour

## graphs correlation with Participation Rate Cotnainer

l = sns.jointplot(PContainer,CollectorIncentives, size=5)
l.set_axis_labels("ParticipationRateContainer", "CollectorIncentives")

g = sns.jointplot(PContainer,ContainerIncentives, size=5)
g.set_axis_labels("ParticipationRateContainer", "ContainerIncentives")
```

```
        h = sns.jointplot(PContainer,WAttitude, size=5)
        h.set_axis_labels("ParticipationRateContainer", "WeightOfAttitude")

        i = sns.jointplot(PContainer,WSocialNorm, size=5)
        i.set_axis_labels("ParticipationRateContainer", "WeightOfSocialNorm")

        j = sns.jointplot(PContainer,WBehaviouralControl, size=5)
        j.set_axis_labels("ParticipationRateContainer", "WeightOfPerceivedBehaviour


        #plt.show()
```

In [ ]:
```
        #Prim analysis

        #Mark outcomes as interesting or not.
        def classify(outcomes):
            outcome = outcomes['ParticipationRateLandfill'][:,-1]
            classes = np.zeros(outcome.shape)
            classes[outcome<0.75] = 1 #interesting outcomes are marked with a 1. Th
            return classes

        prim_obj = setup_prim(results, classify, threshold=0.7)
        box1 = prim_obj.find_box()
```

In [ ]:
```
        box1.show_tradeoff() #show boxes of the prim analysis
        plt.show()

        box1.show_tradeoff()  #same graph, but interactive
        mpld3.display()
```

In [ ]:
```
        box1.inspect(19, style = "graph") #inspect interesting box


        box1.show_ppt()
        box1.write_ppt_to_stdout() #show all boxes

        prim_obj.show_boxes()
        plt.show()
```

In [ ]:
```
        #Same prim analysis, but with other condition for interesting points
        def classify(outcomes):
            outcome = outcomes['ParticipationRateLandfill'][:,-1]
            classes = np.zeros(outcome.shape)
            classes[outcome>0.95] = 1
            return classes

        prim_obj = setup_prim(results, classify, threshold=0.7)
        box1 = prim_obj.find_box()
```

```
In [ ]: box1.show_tradeoff()
        plt.show()

        box1.show_tradeoff()
        mpld3.display()

In [ ]: box1.inspect(1, style = "graph")

In [ ]: #optional plot to clarify PRIM

        from ema_workbench.analysis.plotting_util import BOXPLOT

        indices = box1.yi

        all_indices = np.arange(experiments.shape[0])
        logical = np.ones((all_indices.shape[0],), dtype=np.bool)
        logical[indices] = 0
        out_box = all_indices[logical]

        gs = {}
        gs['in box'] = indices
        gs['out box'] = out_box
        # gs['all'] = all_indices

        envelopes(results, outcomes_to_show=['ParticipationRateLandfill'], group_by
        plt.show()

In [ ]:
```