Residential Rooftop PV Example

Set up.

One only needs to execute the following line once, in order to make sure recent enough packages are installed.

In [ ]:
```
#!pip install 'numpy>=1.17.2' 'pandas>=0.25.1'
```

Import packages.

In [1]:
```
import os
import sys
sys.path.insert(0, os.path.abspath("../src"))
```

In [2]:
```
import numpy                as np
import matplotlib.pyplot as pl
import pandas               as pd
import re                   as re
import scipy.stats          as st
import seaborn              as sb

# The `tyche` package is located at <https://github.com/NREL/portfoli
o/tree/master/production-function/framework/src/tyche/>.
import tyche                as ty

from copy import deepcopy
```

Scenario analyses.

Load data.

The data are stored in a set of tab-separated value files in a folder.

In [3]:
```
designs = ty.Designs("../data/residential_pv")
```

Compile the production and metric functions for each technology in the dataset.

In [4]:
```
designs.compile()
```

Examine the data.

The `functions` table specifies where the Python code for each technology resides.

In [5]:
```
designs.functions
```

Out[5]:

	Style	Module	Capital	Fixed	Production	Metrics	Notes
Technology							
Residential PV	numpy	residential_pv	capital_cost	fixed_cost	production	metrics	

Right now, only the style `numpy` is supported.

The `indices` table defines the subscripts for variables.

In [6]:
```
designs.indices.sort_values(["Technology", "Type", "Offset"])
```

Out[6]:

			Offset	Description	Notes
Technology	**Type**	**Index**			
		Module	0	system module	
	Capital	Inverter	1	system inverters	
		BoS	2	balance of system	
Residential PV	Fixed	System	0	whole system	
	Input	NaN	0	no inputs	
	Metric	LCOE	0	levelized cost of energy	
	Output	Electricity	0	electricity generated	

The `designs` table contains the cost, input, efficiency, and price data for a scenario.

In [7]:
```
designs.designs.xs("2015 Actual", level="Scenario", drop_level=False)
```

Out[7]:

Technology	Scenario	Variable	Index	Value	Units	Notes
Residential PV	2015 Actual	Input	NaN	0	1	no inputs
		Input efficiency	NaN	1	1	no inputs
		Input price	NaN	0	1	no inputs
		Lifetime	BoS	1	system-lifetime	per-lifetime computations
			Inverter	1	system-lifetime	per-lifetime computations
			Module	1	system-lifetime	per-lifetime computations
		Output efficiency	Electricity	1	W/W	see parameter table for individual efficiencies
		Output price	Electricity	0	$/kWh	not tracking electricity price
		Scale	NaN	1	system/system	no scaling

The `parameters` table contains additional techno-economic parameters for each technology.

In [8]:
```
designs.parameters.xs("2015 Actual", level="Scenario", drop_level=False).sort_values(["Technology", "Scenario", "Offset"])
```

Out[8]:

Technology	Scenario	Parameter	Offset	Value	Units	Notes
Residential PV	2015 Actual	Discount Rate	0	0.07	1/year	DR
		Insolation	1	1000	W/m^2	INS
		System Size	2	36	m^2	SSZ
		Module Capital	3	st.triang(0.5, loc=110, scale=0.11)	$/m^2	MCC
		Module Lifetime	4	st.triang(0.5, loc=25, scale=0.0025)	yr	MLT
		Module Efficiency	5	st.triang(0.5, loc=0.16, scale=1.6e-5)	1	MEF
		Module Aperture	6	st.triang(0.5, loc=0.9, scale=9e-5)	1	MAP
		Module O&M Fixed	7	st.triang(0.5, loc=20, scale=0.002)	$/kWyr	MOM
		Module Degradation	8	st.triang(0.5, loc=0.0075, scale=7.5e-7)	1/yr	MDR
		Location Capacity Factor	9	st.triang(0.5, loc=0.2, scale=2e-5)	1	MCF
		Module Soiling Loss	10	st.triang(0.5, loc=0.05, scale=5e-6)	1	MSL
		Inverter Capital	11	st.triang(0.5, loc=0.3, scale=3e-5)	$/W	ICC
		Inverter Lifetime	12	st.triang(0.5, loc=16, scale=0.0016)	yr	ILT
		Inverter Replacement	13	st.triang(0.5, loc=0.5, scale=5e-5)	1	IRC
		Inverter Efficiency	14	st.triang(0.5, loc=0.9, scale=9e-5)	1	IEF
		DC-to-AC Ratio	15	st.triang(0.5, loc=1.4, scale=0.00014)	1	IDC
		Hardware Capital	16	st.triang(0.5, loc=80, scale=0.008)	$/m^2	BCC
		Direct Labor	17	st.triang(0.5, loc=2000, scale=0.2)	$/system	BLR
		Permitting	18	st.triang(0.5, loc=600, scale=0.06)	$/system	BPR
		Customer Acquisition	19	st.triang(0.5, loc=2000, scale=0.2)	$/system	BCA
		Installer Overhead & Profit	20	st.triang(0.5, loc=0.35, scale=3.5e-5)	1	BOH

The `results` table specifies the units of measure for results of computations.

In [9]:
```
designs.results
```

Out[9]:

Technology	Variable	Index	Units	Notes
Residential PV	Cost	Cost	$/system	
	Metric	LCOE	$/kWh	
	Output	Electricity	kWh	

Here is the source code for the computations.

In [10]:
```
!cat ../src/technology/residential_pv.py
```

```
# Residential PV

# All of the computations must be vectorized, so use `numpy`.
import numpy as np


# Discount at a rate for a time.
def discount(rate, time):
  return 1 / (1 + rate)**time


# Net present value of constant cash flow.
def npv(rate, time):
  return (1 - 1 / (1 + rate)**(time + 1)) / (1 - 1 / (1 + rate))


# Capital-cost function.
def capital_cost(scale, parameter):

  # For readability, copy the parameter vectors to named variables.
  dr  = parameter[ 0]
  ins = parameter[ 1]
  ssz = parameter[ 2]
  mcc = parameter[ 3]
  mlt = parameter[ 4]
  mef = parameter[ 5]
  icc = parameter[11]
  ilt = parameter[12]
  irc = parameter[13]
  bcc = parameter[16]
  blr = parameter[17]
  bpr = parameter[18]
  bca = parameter[19]
  boh = parameter[20]

  # System module capital cost.
  smcxa = ssz * mcc

  # System inverter capital cost.
  sicxa = ins * ssz * mef * icc

  # One inverter replacement.
  rsicxa1 = (mlt > ilt) * (mlt < 2 * ilt) * \
            (discount(dr, ilt) - (2 * ilt - mlt) / ilt * discount(dr,
mlt))

  # Two inverter replacements.
  rsicxa2 = (mlt > 2 * ilt) * (mlt < 3 * ilt) * \
            (discount(dr, ilt) + discount(dr, 2 * ilt) - (3 * ilt - m
lt) / ilt * discount(dr, mlt))

  # Capital cost of all inverters.
  # FIXME: Generalize to an arbitrary number of inverter replacement
s.
  sicxa = sicxa * (1 + irc * (rsicxa1 + rsicxa2))

  # System BOS hardware cost.
```

```
  sbh = bcc * ssz

  # System BOS soft costs for labor, permitting, and customers.
  sbs = blr + bpr + bca

  # System overhead costs.
  soh = boh * (smcxa + sicxa + sbh + sbs)

  # Return the capital costs.
  return np.stack([
    smcxa          , # module
    sicxa          , # inverters
    sbh + sbs + soh, # balance of system
  ])


# Fixed-cost function.
def fixed_cost(scale, parameter):

  # For readability, copy the parameter vectors to named variables.
  dr  = parameter[ 0]
  ins = parameter[ 1]
  ssz = parameter[ 2]
  mlt = parameter[ 4]
  mef = parameter[ 5]
  mom = parameter[ 7]

  # System lifetime overhead costs.
  return np.stack([
    mom * ins / 1000 * ssz * mef * npv(dr, mlt)
  ])


# Production function.
def production(scale, capital, lifetime, fixed, input, parameter):

  # For readability, copy the parameter vectors to named variables.
  ins = parameter[ 1]
  ssz = parameter[ 2]
  mlt = parameter[ 4]
  mef = parameter[ 5]
  map = parameter[ 6]
  mdr = parameter[ 8]
  mcf = parameter[ 9]
  msl = parameter[10]
  ief = parameter[14]

  # System lifetime energy conversion.
  return np.stack([
    ins / 1000 * 24 * 365 * ssz * map * mcf * mef * ief * (1 - msl) *
npv(mdr / (1 - mdr), mlt)
  ])


# Metrics function.
def metrics(scale, capital, lifetime, fixed, input_raw, input, output
_raw, output, cost, parameter):
```

```
    # Levelized cost of energy.
    return np.stack([
      cost / output[0]
    ])
```

Evaluate the scenarios in the dataset.

In [11]:
```
scenario_results = designs.evaluate_scenarios(sample_count=500)
```

In [12]:
```
scenario_results
```

Out[12]:

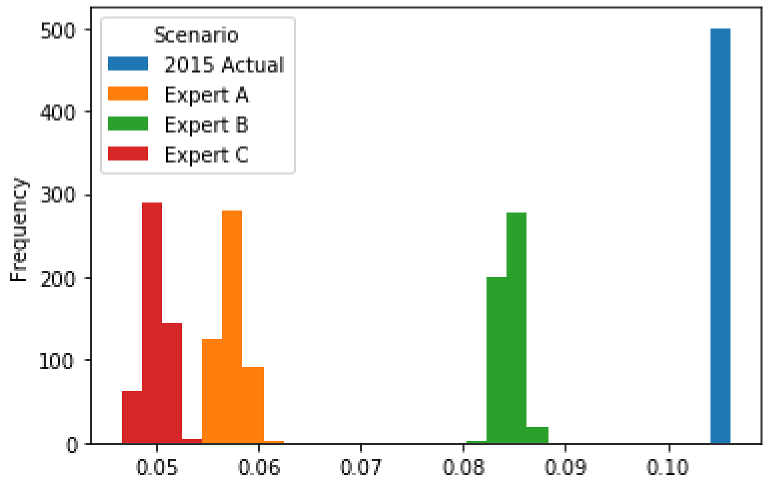
					Value	Units
Technology	**Scenario**	**Sample**	**Variable**	**Index**		
			Cost	Cost	19537.416166	$/system
		1	Metric	LCOE	0.106118	$/kWh
	2015 Actual		Output	Electricity	184110.179812	kWh
			Cost	Cost	19539.145448	$/system
		2	Metric	LCOE	0.106125	$/kWh
Residential PV	...	...	...	...	...	...
			Metric	LCOE	0.049980	$/kWh
		499	Output	Electricity	292615.533400	kWh
	Expert C		Cost	Cost	14645.735937	$/system
		500	Metric	LCOE	0.049193	$/kWh
			Output	Electricity	297720.342805	kWh

6000 rows × 2 columns

Plot the results.

In [13]:
```
expert_results = scenario_results[["Value"]].xs(
    "LCOE", level="Index"
).rename(
    columns={"Value" : "LCOE [$/kWh]"}
).unstack(
    ["Scenario"]
).xs("LCOE [$/kWh]", axis=1, drop_level=True).reset_index(drop=True)
expert_results.plot.hist(bins=30)
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd517a0c470>
```



Make tornado plots for Expert A.

Remember base case LCOE.

In [14]:
```
base_lcoe = scenario_results.xs(["2015 Actual", "LCOE"], level=["Scen
ario", "Index"])[["Value"]].agg(np.mean)[0]
base_lcoe
```

Out[14]:
```
0.10613269974604357
```

Define the factors.

In [15]:
```
tornado_factors = [
    "MCC", "MLT", "MEF", "MAP", "MOM",
    "MDR", "ICC", "ILT", "IRC", "IEF",
    "BCC", "BLR", "BPR", "BCA", "BOH",
]
```

Add the scenarios to the design.

In [16]:
```
design_2015_actual    = designs.designs.xs   ("2015 Actual", level="S
cenario")
parameter_2015_actual = designs.parameters.xs("2015 Actual", level="S
cenario")
parameter_expert_a    = designs.parameters.xs("Expert A"   , level="S
cenario")
for factor in tornado_factors:
    scenario_new = factor
    design_new = design_2015_actual.copy()
    design_new["Scenario"] = scenario_new
    designs.designs = designs.designs.append(design_new.reset_index()
.set_index(["Technology", "Scenario", "Variable", "Index"]))
    parameter_new = pd.concat([
        parameter_2015_actual[parameter_2015_actual["Notes"] != facto
r],
        parameter_expert_a   [parameter_expert_a   ["Notes"] == facto
r],
    ])
    parameter_new["Scenario"] = factor
    designs.parameters = designs.parameters.append(parameter_new.rese
t_index().set_index(["Technology", "Scenario", "Parameter"]))
```

Recompile the design.

In [17]:
```
designs.compile()
```

Compute the results.

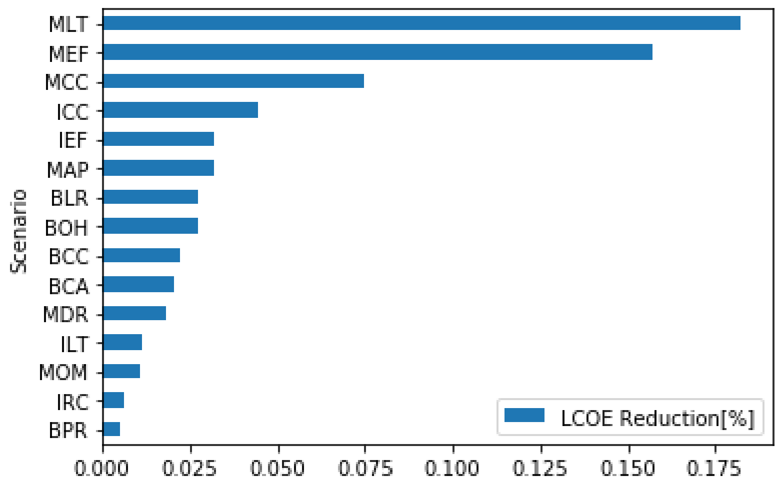
In [18]:
```
scenario_results = designs.evaluate_scenarios(sample_count=500)
scenario_results.shape
```

Out[18]:
```
(28500, 2)
```

Make the tornado plot.

In [19]:
```
tornado_results = scenario_results[["Value"]].xs(
    "LCOE", level="Index"
).rename(
    columns={"Value" : "LCOE [$/kWh]"}
).reset_index(
    ["Technology", "Sample", "Variable"], drop=True
).groupby("Scenario").agg(np.min).drop(["2015 Actual", "Expert A", "E
xpert B", "Expert C"])
tornado_results["LCOE Reduction[%]"] = 1 - tornado_results["LCOE [$/k
Wh]"] / base_lcoe
tornado_results[["LCOE Reduction[%]"]].sort_values("LCOE Reduction
[%]", ascending=True).plot.barh()
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd517ac5240>



Look at the uncertainties.

In [20]:

```
sb.boxplot(
    data = scenario_results[["Value"]].xs(
        "LCOE", level="Index"
    ).rename(
        columns={"Value" : "LCOE [$/kWh]"}
    ).reset_index(
        ["Technology", "Sample", "Variable"], drop=True
    ).drop(["2015 Actual", "Expert A", "Expert B", "Expert C"]).reset
_index().sort_values("LCOE [$/kWh]"),
    x = "Scenario",
    y = "LCOE [$/kWh]"
)
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd5172d4a20>