1 Resource Allocation LP Problem: Blue Ridge Hot Tubs

A popular Python package for creating and solving linear programming problems is PuLP.

Here are links to the documentation, which has some good information in it.

https://coin-or.github.io/pulp/main/index.html (https://coin-or.github.io/pulp/main/index.html)

https://pypi.org/project/PuLP/ (https://pypi.org/project/PuLP/)

To install Pulp through a Jupyter Notebook, you can run the shell command with the '!' character.

In [1]:
```
## !pip install pulp
```

In [2]:
```
from pulp import LpMaximize, LpProblem, LpStatus, lpSum, LpVariable
```

Blue Ridge Hot Tubs produces two types of hot tubs: Aqua-Spas & Hydro-Luxes. Howie Jones, owner-operator of the company, needs to decide how many of each type of hot tub to produce during the next production cycle.

Howie buys prefabricated fiberglass hot tub shells from a local supplier and adds the pump and tubing to the shells to create his hot tubs. Howie installs the same type of pump into both hot tub models. He will have only 200 pumps available for the next production rub. In manufacturing the hot tubs, the main difference between the Aqua-Spa and Hydro-Lux models is the amount of tubing and labor required. Each Aqua-Spa requires 9 hours of labor and 12 feet of tubing. Each Hydro-Lux requires 6 hours of labor and 16 feet of tubing.

Howie expects to have 1,566 production labor hours and 2,880 feet of tubing available. Howie earns a profit of $350 on each Aqua-Spa he sells and $300 on each Hydro-Lux he sells.

How many of each model hot tub should Blue Ridge produce in order to maximize profits during the next production cycle?

Let X1 = number of Aqua-Spa, X2 = number of Hydro-Lux models

Maximize 350(X1) + 300(X2)

subject to:
1(X1) + 1(X2) <= 200
9(X1) + 6(X2) <= 1566
12(X1) + 16(X2) <= 2880

In [3]:
```
# Create the model (LpProblem class objects)
model = LpProblem("HotTubs:Resource_Allocation", LpMaximize)
```

In [4]:
```
# Create the decision variables
x1 = LpVariable('Aqua-Spa', lowBound = 0, cat = 'Integer')
x2 = LpVariable('Hyrdo-Lux', lowBound = 0, cat = 'Integer')
```

An expression added to the model becomes the objective function.

In [5]:
```
# Add the objective function to the model
obj_func = 350 * x1 + 300 * x2
model += obj_func
```

Constraints are added to the model with both left hand and right hand sides.

In [6]:
```
# Add the constraints to the model
model += (1 * x1 + 1 * x2 <= 200, "pumps")
model += (9 * x1 + 6 * x2 <= 1566, "labor")
model += (12 * x1 + 16 * x2 <= 2800, "tubing")
```

In [7]:
```
model
```

Out[7]:
```
HotTubs:Resource_Allocation:
MAXIMIZE
350*Aqua_Spa + 300*Hyrdo_Lux + 0
SUBJECT TO
pumps: Aqua_Spa + Hyrdo_Lux <= 200

labor: 9 Aqua_Spa + 6 Hyrdo_Lux <= 1566

tubing: 12 Aqua_Spa + 16 Hyrdo_Lux <= 2800

VARIABLES
0 <= Aqua_Spa Integer
0 <= Hyrdo_Lux Integer
```

In [8]:
```
model.solve()

LpStatus[model.status]
```

Out[8]:
```
'Optimal'
```

In [9]:
```
model.objective.value()
```

Out[9]:
```
66100.0
```

In [10]:
```
## Retrieve optimal values for decision variables
for v in model.variables(): print(f"{v.name}: {v.varValue}")
```

```
Aqua_Spa: 122.0
Hyrdo_Lux: 78.0
```

In [11]:
```
## Retrieve contraint values at optimal solution.  Note that value is relative to
for name, constraint in model.constraints.items(): print(f"{name}: {constraint.va
```

```
pumps: 0.0
labor: 0.0
tubing: -88.0
```

In [12]:
```
model.constraints['tubing'].value() - model.constraints['tubing'].constant
```

Out[12]:
```
2712.0
```

In [13]:
```
model.solver
```

Out[13]:
```
<pulp.apis.coin_api.PULP_CBC_CMD at 0x2134a9b37c0>
```

In [ ]:
```

```