Basics of optimization

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

datacamp

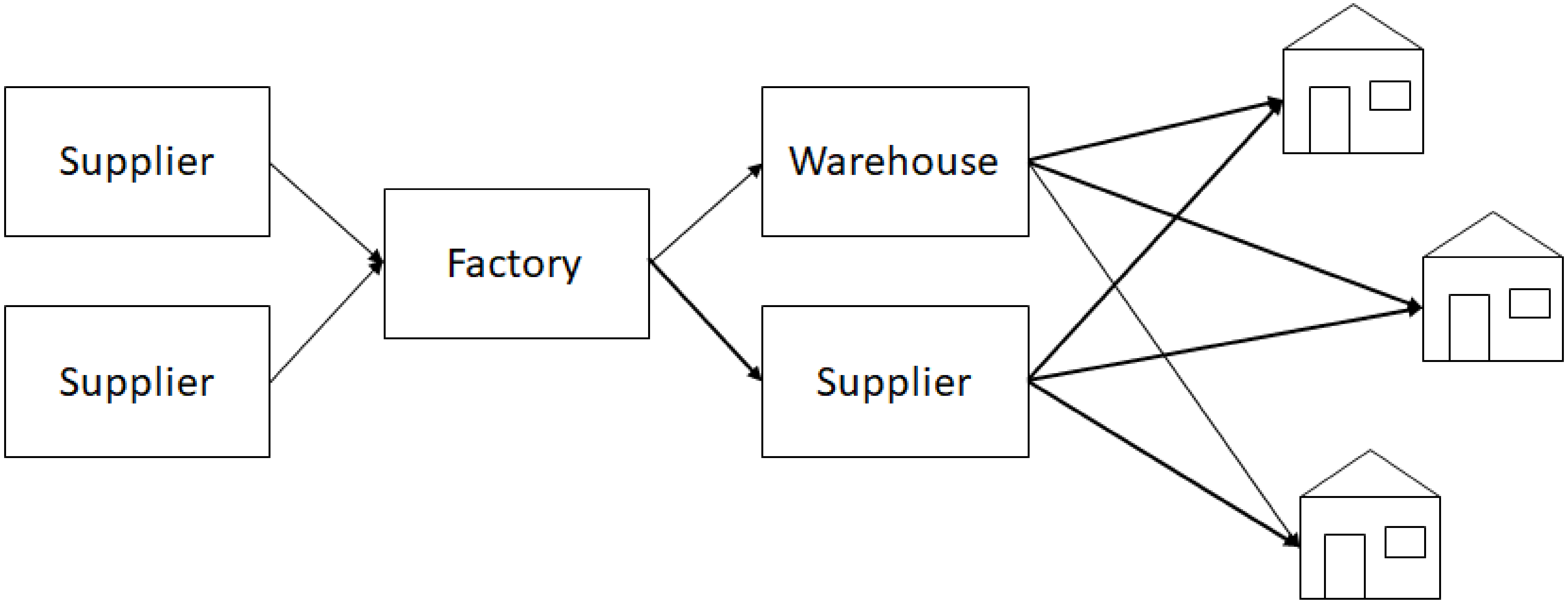
What is a supply chain

- A Supply Chain consist of all parties involved, directly or indirectly, in fulfilling a customer's request.[1]
- Includes:
    - Suppliers
    - Internal Manufacturing
    - Outsourced Logistics Suppliers (i.e. Third Party Suppliers)

[1] Chopra, Sunil, and Peter Meindl. _Supply Chain Management: Strategy, Planning, and Operations._ Pearson Prentice-Hall, 2007.

What is a supply chain optimization

- Involves finding the best path to achieve an objective based on constraints



Crash course in LP

- Linear Programing (LP) is a Powerful Modeling Tool for Optimization
- Optimization method using a mathematical model whose requirements are linear relationships
- There are 3 Basic Components in LP:
  - Decision Variables - ***what you can control***
  - Objective Function - ***math expression that uses variables to express goal***
  - Constraints - ***math expression that describe the limits of a solutions***

Introductory example

Use LP to decide on an exercise routine to burn as many calories as possible.

	Pushup	Running
Minutes	0.2 per pushup	10 per mile
Calories	3 per pushup	130 per mile

Constraint - only 10 minutes to exercise

Basic components of an LP

Decision Variables - What we can control:

- Number of Pushups & Number of Miles Ran

Objective Function - Math expression that uses variables to express goal:

- Max (3 * Number of Pushups + 130 * Number of Miles)

Constraints - Math expression that describe the limits of a solutions:

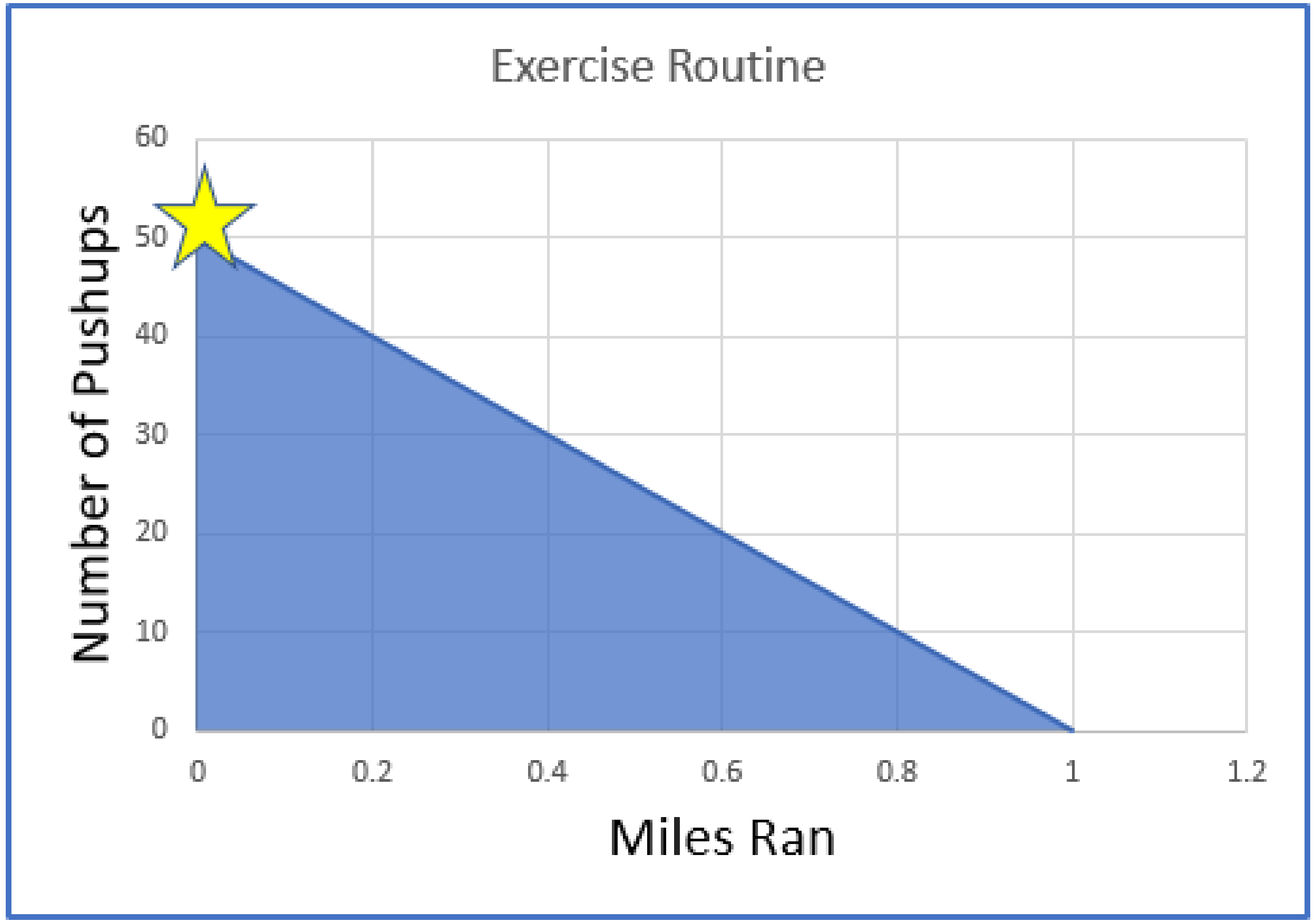
- 0.2 * Number of Pushups + 10 * Number of Miles ≤ 10
- Number of Pushups ≥ 0
- Number of Miles ≥ 0

Example solution

Optimal Solution:

- 50 Pushups
- 0 Miles Ran

Calories Burned: 150



LP vs IP vs MIP

Terms	Decision Variables
Linear Programing (LP)	Only Continuous
Integer Programing (IP)	Only Discrete or Integers
Mixed Integer Programing (MIP)	Mix of Continuous and Discrete

Summary

- Defined Supply Chain Optimization
- Defined Linear Programing and Basic Components
  - Decision Variables
  - Objective Function
  - Constraints
- Defined LP vs IP vs MIP

Let's practice!

SUPPLY CHAIN ANALYTICS IN PYTHON

Basics of PuLP modeling

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

datacamp

What is PuLP

- `PuLP` is a modeling framework for Linear (LP) and Integer Programing (IP) problems written in Python
- Maintained by COIN-OR Foundation (Computational Infrastructure for Operations Research)
- `PuLP` interfaces with Solvers
  - `CPLEX`
  - `COIN`
  - `Gurobi`
  - etc...

PuLP example – resource scheduling

- Consultant for boutique cake bakery that sell 2 types of cakes
- 30 day month
- There is:
  - 1 oven
  - 2 bakers
  - 1 packaging packer – only works 22 days

PuLP example – resource scheduling

- Different resource needs for the 2 types of cakes:

	Cake A	Cake B
Oven	0.5 days	1 day
Bakers	1 day	2.5 days
Packers	1 day	2 days

-

	Cake A	Cake B
Profit	$20.00	$40.00

PuLP example – resource scheduling

- Objective is to Maximize Profit
  - Profit = 20*A + 40*B
- Subject to:
  - $A \geq 0$
  - $B \geq 0$
  - $0.5A + 1B \leq 30$
  - $1A + 2.5B \leq 60$
  - $1A + 2B \leq 22$

Common modeling process for PuLP

1. Initialize Model
2. Define Decision Variables
3. Define the Objective Function
4. Define the Constraints
5. Solve Model

Initializing model - LpProblem()

```
LpProblem(name='NoName', sense=LpMinimize)
```

- `name` = Name of the problem used in the output .lp file, *i.e. "My LP Problem"*
- `sense` = Maximize or minimize the objective function
  - Minimize = `LpMinimize` *(default)*
  - Maximize = `LpMaximize`

PuLP example – resource scheduling

1. **Initialize Model**

```
from pulp import *


# Initialize Class
model = LpProblem("Maximize Bakery Profits", LpMaximize)
```

Define decision variables - LpVariable()

```
LpVariable(name, lowBound=None, upBound=None, cat='Continuous', e=None)
```

- `name` = Name of the variable used in the output .lp file
- `lowBound` = Lower bound
- `upBound` = Upper bound
- `cat` = The type of variable this is
  - Integer
  - Binary
  - Continuous *(default)*
- `e` = Used for column based modeling

PuLP example – resource scheduling

1. Initialize Class
2. **Define Variables**

```
# Define Decision Variables
A = LpVariable('A', lowBound=0, cat='Integer')
B = LpVariable('B', lowBound=0, cat='Integer')
```

PuLP example – resource scheduling

1. Initialize Class
2. Define Variables
3. **Define Objective Function**

```
# Define Objective Function
model += 20 * A + 40 * B
```

PuLP example – resource scheduling

1. Initialize Class
2. Define Variables
3. Define Objective Function
4. **Define Constraints**

```
# Define Constraints
model += 0.5 * A + 1 * B <= 30
model += 1 * A + 2.5 * B <= 60
model += 1 * A + 2 * B <= 22
```

PuLP example – resource scheduling

1. Initialize Class
2. Define Variables
3. Define Objective Function
4. Define Constraints
5. **Solve Model**

```
# Solve Model
model.solve()
print("Produce {} Cake A".format(A.varValue))
print("Produce {} Cake B".format(B.varValue))
```

PuLP example – resource scheduling

```python
from pulp import *

# Initialize Class
model = LpProblem("Maximize Bakery Profits",
                  LpMaximize)

# Define Decision Variables
A = LpVariable('A', lowBound=0,
                cat='Integer')
B = LpVariable('B', lowBound=0,
                cat='Integer')

# Define Objective Function
model += 20 * A + 40 * B
```

```python
# Define Constraints
model += 0.5 * A + 1 * B <= 30
model += 1 * A + 2.5 * B <= 60
model += 1 * A + 2 * B <= 22

# Solve Model
model.solve()
print("Produce {} Cake A".format(A.varValue))
print("Produce {} Cake B".format(B.varValue))
```

Summary

- PuLP is a Python LP / IP modeler
- Reviewed 5 Steps of PuLP modeling process
  1. Initialize Model
  2. Define Decision Variables
  3. Define the Objective Function
  4. Define the Constraints
  5. Solve Model
- Completed Resource Scheduling Example

Let's practice!

SUPPLY CHAIN ANALYTICS IN PYTHON

Using IpSum

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

datacamp

Moving from simple to complex

Simple Bakery Example

```
# Define Decision Variables
A = LpVariable('A', lowBound=0, cat='Integer')
B = LpVariable('B', lowBound=0, cat='Integer')
```

More Complex Bakery Example

```
# Define Decision Variables
A = LpVariable('A', lowBound=0, cat='Integer')
B = LpVariable('B', lowBound=0, cat='Integer')
C = LpVariable('C', lowBound=0, cat='Integer')
D = LpVariable('D', lowBound=0, cat='Integer')
E = LpVariable('E', lowBound=0, cat='Integer')
F = LpVariable('F', lowBound=0, cat='Integer')
```

Moving from simple to complex

Objective Function of Complex Bakery Example

```
# Define Objective Function
model += 20*A + 40*B + 33*C + 14*D + 6*E + 60*F
```

Need method to scale

$$z = X1 + X2 + X3 + \cdots + Xk$$

Using lpSum()

```
lpSum(vector)
```

- `vector` = A list of linear expressions

Therefore ...

```
# Define Objective Function
model += 20*A + 40*B + 33*C + 14*D + 6*E + 60*F
```

Equivalent to ...

```
# Define Objective Function
var_list = [20*A, 40*B, 33*C, 14*D, 6*E, 60*F]
model += lpSum(var_list)
```

lpSum with list comprehension

```
# Define Objective Function
cake_types = ["A", "B", "C", "D", "E", "F"]
profit_by_cake = {"A":20, "B":40, "C":33, "D":14, "E":6, "F":60}
var_dict = {"A":A, "B":B, "C":C, "D":D, "E":E, "F":F}

model += lpSum([profit_by_cake[type] * var_dict[type]
                for type in cake_types])
```

Summary

- Need way to sum many variables
- `lpSum()`
- Used in list comprehension

Practice time!

SUPPLY CHAIN ANALYTICS IN PYTHON

LpVariable dictionary function

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

datacamp

Moving from simple to complex

Complex Bakery Example

```
# Define Decision Variables
A = LpVariable('A', lowBound=0, cat='Integer')
B = LpVariable('B', lowBound=0, cat='Integer')
C = LpVariable('C', lowBound=0, cat='Integer')
D = LpVariable('D', lowBound=0, cat='Integer')
E = LpVariable('E', lowBound=0, cat='Integer')
F = LpVariable('F', lowBound=0, cat='Integer')
```

```
# Define Objective Function
var_dict = {"A":A, "B":B, "C":C, "D":D, "E":E, "F":F}

# Define Objective Function
model += lpSum([profit_by_cake[type] * var_dict[type] for type in cake_types])
```

Using LpVariable.dicts()

```
LpVariable(name, indexs, lowBound=None, upBound=None, cat='Continuous')
```

- `name` = The prefix to the name of each LP variable created
- `indexs` = A list of strings of the keys to the dictionary of LP variables
- `lowBound` = Lower bound
- `upBound` = Upper bound
- `cat` = The type of variable this is
  - Integer
  - Binary
  - Continuous *(default)*

LpVariable.dicts() with list comprehension

- `LpVariable.dicts()` often used with Python's list comprehension

Transportation Optimization

```
# Define Decision Variables
customers = ['East','South','Midwest','West']
warehouse = ['New York','Atlanta']
transport = LpVariable.dicts("route", [(w,c) for w in warehouse for c in customers],
                             lowBound=0, cat='Integer')
# Define Objective
model += lpSum([cost[(w,c)]*transport[(w,c)] for w in warehouse for c in customers])
```

Summary

- Creating many LP variables for complex problems
- `LpVariable.dicts()`
- Used with list comprehension

Now you try it out

SUPPLY CHAIN ANALYTICS IN PYTHON

Example of a scheduling problem

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

datacamp

Expected Demand

Day of Week	Drivers Needed
0 = Monday	11
1= Tuesday	14
2 = Wednesday	23
3 = Thursday	21
4 = Friday	20
5 = Saturday	15
6 = Sunday	8

Question:

- How many drivers, in total, do we need to hire?

Constraint:

- Each driver works for 5 consecutive days, followed by 2 days off, repeated weekly

Step	Definition
Decision Var	X_i = the number of drivers working on day _i_
Objective	*minimize* $z = X_0 + X_1 + X_2 + X_3 + X_4 + X_5 + X_6$
Subject to	$X_0 \geq 11$
	$X_1 \geq 14$
	$X_2 \geq 23$
	$X_3 \geq 21$
	$X_4 \geq 20$
	$X_i \geq 0$ (i = 0, ..., 6)

Step	Definition
Decision Var	X_i = the number of drivers working on day _i_
Objective	minimize z = $X_0 + X_1 + X_2 + X_3 + X_4 + X_5 + X_6$
Subject to	$X_0 + X_3 + X_4 + X_5 + X_6 \geq 11$
	$X_0 + X_1 + X_4 + X_5 + X_6 \geq 14$
	$X_0 + X_1 + X_2 + X_3 + X_6 \geq 23$
	$X_0 + X_1 + X_2 + X_3 + X_4 \geq 21$
	$X_1 + X_2 + X_3 + X_4 + X_5 \geq 15$
	X_i ? 0 (i = 0, ..., 6)

Coding example

```
# Initialize Class
model = LpProblem("Minimize Staffing",
                  LpMinimize)
days = list(range(7))

# Define Decision Variables
x = LpVariable.dicts('staff_', days,
                     lowBound=0, cat='Integer')

# Define Objective
model += lpSum([x[i] for i in days])
```

```
# Define Constraints
model += x[0] + x[3] + x[4] + x[5] + x[6] >= 11
model += x[0] + x[1] + x[4] + x[5] + x[6] >= 14
model += x[0] + x[1] + x[2] + x[5] + x[6] >= 23
model += x[0] + x[1] + x[2] + x[3] + x[6] >= 21
model += x[0] + x[1] + x[2] + x[3] + x[4] >= 20
model += x[1] + x[2] + x[3] + x[4] + x[5] >= 15
model += x[2] + x[3] + x[4] + x[5] + x[6] >= 8

# Solve Model
model.solve()
```

Summary

- Our initial variables did not work
- Decision variables to incorporate some of the constraints

Practice time!

SUPPLY CHAIN ANALYTICS IN PYTHON

Capacitated plant location - case study P1

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

Context

Multiple options to meet regional product demand

Option	Pro	Con
Small manufacturing facilities within region	Low transportation costs, few to no tariffs or duties	Overall network may have excess capacity, cannot take advantage economies of scale
A few large manufacturing plants and ship product to region	Economies of scale	Higher transportation, higher tariffs and duties

Capacitated plant location model

- Capacitated Plant Location Model[1]
- The goal is to optimize global Supply Chain network
  - Meet regional demand at the lowest cost
  - Determine regional production of a product

[1] Chopra, Sunil, and Peter Meindl. _Supply Chain Management: Strategy, Planning, and Operations._ Pearson Prentice-Hall, 2007.

Capacitated plant location model

Modeling

- Production at regional facilities
  - Two plant sizes (low / high)
- Exporting production to other regions
- Production facilities open / close

Decision variables

What we can control:

- x_{ij} = quantity produced at location **_i_** and shipped to **_j_**
- y_{is} = 1 if the plant at location **_i_** of capacity **_s_** is open, 0 if closed
  - s = ***low*** or ***high*** capacity plant

Objective function

Minimize $z = \sum_{i=1}^{n}(f_{is}y_{is}) + \sum_{i=1}^{n}\sum_{i=1}^{m}(c_{ij}x_{ij})$

- c_{ij} = cost of producing and shipping from plant **_i_** to region **_j_**
- f_{is} = fixed cost of keeping plant **_i_** of capacity **_s_** open
- n = number of production facilities
- m = number of markets or regional demand points

```python
from pulp import *

# Initialize Class
model = LpProblem("Capacitated Plant Location Model", LpMinimize)

# Define Decision Variables
loc = ['A', 'B', 'C', 'D', 'E']
size = ['Low_Cap','High_Cap']
x = LpVariable.dicts("production",[(i,j) for i in loc for j in loc],
                      lowBound=0, upBound=None, cat='Continous')
y = LpVariable.dicts("plant",[(i,s) for s in size for i in loc], cat='Binary')
# Define objective function
model += (lpSum([fix_cost.loc[i,s]*y[(i,s)] for s in size for i in loc])
          + lpSum([var_cost.loc[i,j]*x[(i,j)] for i in loc for j in loc]))
```

Summary

Capacitated Plant Location Model:

- Finds a balance between the number of production facilities
- Model decision variables:
  - Quantity of production in a region and exported
  - High or low capacity facilities open or closed
- Reviewed objective function
  - Sums variable and fixed production costs
- Reviewed code example

Review time

SUPPLY CHAIN ANALYTICS IN PYTHON

Logical constraints

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

Example problem

Maximum Weight 20,000 lbs

Product	Weight (lbs)	Profitability ($US)
A	12,800	77,878
B	10,900	82,713
C	11,400	82,728
D	2,100	68,423
E	11,300	84,119
F	2,300	77,765

- Select most profitable product to ship without exceeding weight limit
- Decision Variables:
  - X_i = 1 if product _i_ is selected else 0
- Objective:
  - Maximize z = $\sum$ Profitability$_i X_i$
- Constraint:
  - $\sum$ Weight$_i X_i$ < 20,0000

```python
prod = ['A', 'B', 'C', 'D', 'E', 'F']
weight = {'A':12800, 'B':10900, 'C':11400, 'D':2100, 'E':11300, 'F':2300}
prof = {'A':77878, 'B':82713, 'C':82728, 'D':68423, 'E':84119, 'F':77765}
# Initialize Class
model = LpProblem("Loading Truck Problem", LpMaximize)

# Define Decision Variables
x = LpVariable.dicts('ship_', prod, cat='Binary')
# Define Objective
model += lpSum([prof[i]*x[i] for i in prod])

# Define Constraint
model += lpSum([weight[i]*x[i] for i in prod]) <= 20000
# Solve Model
model.solve()
for i in prod: print("{} status {}".format(i, x[i].varValue))
```

Example result

Maximum Weight 20,000 lbs

Product	Ship or Not
A	No
B	No
C	No
D	Yes
E	Yes
F	Yes

Result

- Profitability: $230,307
- Weight of Products: 15,700 lbs

Logical constraint example 1

Either product E is selected or product D is selected, but not both.

- X_E = 1 if product _i_ is selected else 0
- X_D = 1 if product _i_ is selected else 0
- Constraint
  - $X_E + X_D \leq 1$

Code example - logical constraint example 1

```
model += x['E'] + x['D'] <= 1
```

```
prod = ['A', 'B', 'C', 'D', 'E', 'F']
weight = {'A':12800, 'B':10900, 'C':11400,
          'D':2100, 'E':11300, 'F':2300}
prof = {'A':77878, 'B':82713, 'C':82728,
        'D':68423, 'E':84119, 'F':77765}

# Initialize Class
model = LpProblem("Loading Truck Problem",
                  LpMaximize)

# Define Decision Variables
x = LpVariable.dicts('ship_', prod,
                     cat='Binary')
```

```
# Define Objective
model += lpSum([prof[i]*x[i] for i in prod])

# Define Constraint
model +=
  lpSum([weight[i]*x[i] for i in prod]) <= 20000
model += x['E'] + x['D'] <= 1

# Solve Model
model.solve()
for i in prod:
  print("{} status {}".format(i, x[i].varValue))
```

Logical constraint 1 example result

Maximum Weight 20,000 lbs

Product	Ship or Not
A	No
B	No
C	Yes
D	Yes
E	No
F	Yes

Result

- Profitability: $228,916
- Weight of Products: 15,800 lbs

Logical constraint example 2

If product D is selected then product B must also be selected.

- X_D = 1 if product _i_ is selected else 0
- X_B = 1 if product _i_ is selected else 0
- Constraint
  - $X_D \leq X_B$

Code example - logical constraint example 2

```
model += x['D'] <= x['B']
prod = ['A', 'B', 'C', 'D', 'E', 'F']
weight = {'A':12800, 'B':10900, 'C':11400,
          'D':2100, 'E':11300, 'F':2300}
prof = {'A':77878, 'B':82713, 'C':82728,
        'D':68423, 'E':84119, 'F':77765}

# Initialize Class
model = LpProblem("Loading Truck Problem",
                  LpMaximize)

# Define Decision Variables
x = LpVariable.dicts('ship_', prod,
                     cat='Binary')
```

```
# Define Objective
model += lpSum([prof[i]*x[i] for i in prod])

# Define Constraint
model +=
 lpSum([weight[i]*x[i] for i in prod]) <= 20000
model += x['D'] <= x['B']

# Solve Model
model.solve()
for i in prod:
  print("{} status {}".format(i, x[i].varValue))
```

Logical constraint 2 example result

Maximum Weight 20,000 lbs

Product	Ship or Not
A	No
B	Yes
C	No
D	Yes
E	No
F	Yes

Result

- Profitability: $228,901
- Weight of Products: 15,300 lbs

Other logical constraints

Logical Constraint	Constraint
If item _i_ is selected, then item _j_ is also selected.	$x_i - x_j \le 0$
Either item _i_ is selected or item _j_ is selected, but not both.	$x_i + x_j = 1$
If item _i_ is selected, then item _j_ is not selected.	$x_i - x_j \le 1$
If item _i_ is not selected, then item _j_ is not selected.	$-x_i + x_j \le 0$
At most one of items _i_, _j_, and _k_ are selected.	$x_i + x_j + x_k \le 1$

[1] James Orlin, and Ebrahim Nasrabadi. 15.053 Optimization Methods in Management Science. Spring 2013. Massachusetts Institute of Technology: MIT OpenCourseWare. License: Creative Commons BY-NC-SA.

Summary

- Reviewed examples of logical constraints
- Listed a table of other logical constraints

Your turn!

SUPPLY CHAIN ANALYTICS IN PYTHON

Common constraint mistakes

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

datacamp

Dependent demand constraint

Context

- Production Plan
- Planning for 2 products *(A, and B)*
- Planning for production over 3 months *(Jan - Mar)*
- Product A is used as an input for production of product B

Constraint Problem

- For each unit of B, we must also have at least 3 units of A

Dependent demand constraint

For each unit of B, we must also have at least 3 units of A

- $3B \leq A$
- $3(2) \leq A$
- $6 \leq A$

Common Mistake:

- $B \leq 3A$
- $3B = A$

Code example

```python
from pulp import *
demand = {'A':[0,0,0],'B':[8,7,6]}
costs = {'A':[20,17,18],'B':[15,16,15]}

# Initialize Model
model = LpProblem("Aggregate Production Planning",
                  LpMinimize)

# Define Variables
time = [0, 1, 2]
prod = ['A', 'B']
X = LpVariable.dicts(
     "prod", [(p, t) for p in prod for t in time],
      lowBound=0, cat="Integer")
```

```python
# Define Objective
model += lpSum([costs[p][t] * X[(p, t)]
                for p in prod for t in time])

# Define Constraint So Production is >= Demand
for p in prod:
    for t in time:
        model += X[(p, t)] >= demand[p][t]
```

Code example continued

```
for t in time:
    model += 3*X[('B',t)] <= X[('A',t)]
```

Extended constraint

For each unit of B, we must also have at least 3 units of A and ***account for direct to customer sells of A***.

- $3B + Demand_A \leq A$

Combination constraint

Context

- Warehouse distribution plan
- 2 warehouses *(WH1, and WH2)*
- We ship 2 products *(A, and B)* from each warehouse
- Warehouse WH1 is small and can either ship 12 A products per a week or 15 B products per a week

Constraint Problem

- What combinations of A, or B can be shipped in 4 weeks?

- 1 week only: $(1/12)A + (1/15)B \leq 1$

Correct Form

- $(1/12)A + (1/15)B$? $\leq$
- $(1/12)(32) + (1/15)(20) \leq 4$
- $(32/12) + (20/15) \leq 4$
- $4 \leq 4$

Common Mistakes

- $12A + 15B \leq 4$
- $(1/12)A + (1/15)B = 4$

```python
from pulp import *
import pandas as pd
demand = pd.read_csv("Warehouse_Constraint_Demand.csv", index_col=['Product'])
costs = pd.read_csv("Warehouse_Constraint_Cost.csv", index_col=['WH','Product'])

# Initialize Model
model = LpProblem("Distribution Planning", LpMinimize)

# Define Variables
wh = ['W1','W2']
prod = ['A', 'B']
cust = ['C1', 'C2', 'C3', 'C4']
X = LpVariable.dicts("ship", [(w, p, c) for c in cust for p in prod for w in wh],
                     lowBound=0, cat="Integer")
```

Code example continued

```
# Define Objective
model += lpSum([X[(w, p, c)]*costs.loc[(w, p), c]
               for c in cust for p in prod for w in wh])

# Define Constraint So Demand Equals Total Shipments
for c in cust:
    for p in prod:
        model += lpSum([X[(w, p, c)] for w in wh]) == demand.loc[p, c]
```

Code example continued

Constraint

```
model += ((1/12) * lpSum([X['W1', 'A', c] for c in cust])
          + (1/15) * lpSum([X['W1', 'B', c] for c in cust])) <= 4
```

Extend constraint

Warehouse WH1 is small and either ship 12 A products per a week, 15 B products per a week, ***or 5 C products per a week***. What combinations of A, B, or C can be shipped in 4 weeks?

- $(1/12)A + (1/15)B + (1/5)C \leq 4$

Summary

- Common Mistakes
  - Dependent constraint
  - Combination selection constraint
- How to extend constraints
- Check constraint by plugging in a value

Let's practice

Capacitated plant location - case study P2

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
SuSupply Chain Analytics Mgr.

Capacitated plant location model

Modeling

- Production at regional facilities
  - Two plant sizes (low / high)
- Exporting production to other regions
- Production facilities open / close

Decision variables

What we can control:

- x_{ij} = quantity produced at location **_i_** and shipped to **_j_**
- y_{is} = 1 if the plant at location **_i_** of capacity **_s_** is open, 0 if closed
    - s = ***low*** or ***high*** capacity plant

Constraints

- Total Production = Total Demand
  - $\sum_{i=1}^{n}$ x_{ij} = D_j for $j = 1, ..., m$
  - n = number of production facilities
  - m = number of markets or regional demand points

Constraints

- Total Production ? Total Production Capacity
  - $\sum_{j=1}^{m} x_{ij}$? $\sum_{s=1} K_{is} y_{is}$
  - K_{is} = potential production capacity of plant **_i_** of size **_s_**

```python
from pulp import *

# Initialize Class
model = LpProblem("Capacitated Plant Location Model", LpMinimize)

# Define Decision Variables
loc = ['A', 'B', 'C', 'D', 'E']
size = ['Low_Cap','High_Cap']
x = LpVariable.dicts("production_", [(i,j) for i in loc for j in loc],
                     lowBound=0, upBound=None, cat='Continuous')
y = LpVariable.dicts("plant_", [(i,s) for s in size for i in loc], cat='Binary')

# Define Objective Function
model += (lpSum([fix_cost.loc[i,s]*y[(i,s)] for s in size for i in loc])
          + lpSum([var_cost.loc[i,j]*x[(i,j)] for i in loc for j in loc]))
```

Code example continued

```
# Define the Constraints
for j in loc:
    model += lpSum([x[(i, j)] for i in loc]) == demand.loc[j,'Dmd']
for i in loc:
    model += lpSum([x[(i, j)] for j in loc]) <= lpSum([cap.loc[i,s]*y[(i,s)]
                                                       for s in size])
```

Summary

Capacitated Plant Location Model:

- Constraints
  - Total Production = Total Demand
  - Total Production ≤ Total Production Capacity

Review time

SUPPLY CHAIN ANALYTICS IN PYTHON

Solve the PuLP model

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

datacamp

Common modeling process for PuLP

1. ~~Initialize Model~~
2. ~~Define Decision Variables~~
3. ~~Define the Objective Function~~
4. ~~Define the Constraints~~
5. Solve Model
   - call the `solve()` method
   - check the status of the solution
   - print optimized decision variables
   - print optimized objective function

Solve model - solve method

```
.solve(solver=None)
```

- `solver` = Optional: the specific solver to be used, defaults to the default solver.

```python
# Initialize, Define Decision Vars., Objective Function, and Constraints
from pulp import *
import pandas as pd
model = LpProblem("Minimize Transportation Costs", LpMinimize)
cust = ['A','B','C']
warehouse = ['W1','W2']
demand = {'A': 1500, 'B': 900, 'C': 800}
costs = {('W1','A'): 232, ('W1','B'): 255, ('W1','C'): 264,
         ('W2','A'): 255, ('W2','B'): 233, ('W2','C'): 250}
ship = LpVariable.dicts("s_", [(w,c) for w in warehouse for c in cust],
                        lowBound=0, cat='Integer')
model += lpSum([costs[(w, c)] * ship[(w, c)] for w in warehouse for c in cust])
for c in cust: model += lpSum([ship[(w, c)] for w in warehouse]) == demand[c]

# Solve Model
model.solve()
```

Solve model - status of the solution

```
LpStatus[model.status]
```

- **Not Solved:** The status prior to solving the problem.
- **Optimal:** An optimal solution has been found.
- **Infeasible:** There are no feasible solutions (e.g. if you set the constraints x ≤ 1 and x ≥ 2).
- **Unbounded:** The object function is not bounded, maximizing or minimizing the objective will tend towards infinity (e.g. if the only constraint was x ≥ 3).
- **Undefined:** The optimal solution may exist but may not have been found.

[1] Keen, Ben Alex. "Linear Programming with Python and PuLP [2] Part 2." _Ben Alex Keen_, 1 Apr. 2016, benalexkeen.com/linear-programming-with-python-and-pulp-part-2/._{{5}}

```python
# Initialize, Define Decision Vars., Objective Function, and Constraints
from pulp import *
import pandas as pd
model = LpProblem("Minimize Transportation Costs", LpMinimize)
cust = ['A','B','C']
warehouse = ['W1','W2']
demand = {'A': 1500, 'B': 900, 'C': 800}
costs = {('W1','A'): 232, ('W1','B'): 255, ('W1','C'): 264,
         ('W2','A'): 255, ('W2','B'): 233, ('W2','C'): 250}
ship = LpVariable.dicts("s_", [(w,c) for w in warehouse for c in cust], lowBound=0, cat='Integer')
model += lpSum([costs[(w, c)] * ship[(w, c)] for w in warehouse for c in cust])
for c in cust: model += lpSum([ship[(w, c)] for w in warehouse]) == demand[c]
# Solve Model
model.solve()
print("Status:", LpStatus[model.status])
```

```
Status: Optimal
```

Print variables to standard output:

```
for v in model.variables():
    print(v.name, "=", v.varValue)
```

Pandas data structure:

```
o = [{A:ship[(w,'A')].varValue, B:ship[(w,'B')].varValue, C:ship[(w,'C')].varValue}
     for w in warehouse]
print(pd.DataFrame(o, index=warehouse))
```

- loop model variables
- store values in a pandas DataFrame

```python
# Solve Model
model.solve()
print(LpStatus[model.status])
o = [{A:ship[(w,'A')].varValue, B:ship[(w,'B')].varValue, C:ship[(w,'C')].varValue}
     for w in warehouse]
print(pd.DataFrame(o, index=warehouse))
```

Output:

```
Status: Optimal
|       |A      |B      |C      |
|:------|:------|:------|:------|
|W1     |1500.0 |0.0    |0.0    |
|W2     |0.0    |900.0  |800.0  |
```

Solve model - optimized objective function

Print the value of optimized objective function:

```
print("Objective = ", value(model.objective))
```

```python
# Solve Model
model.solve()
print(LpStatus[model.status])
output = []
for w in warehouse: t = [ship[(w,c)].varValue for c in cust] output.append(t)
opd = pd.DataFrame.from_records(output, index=warehouse, columns=cust)
print(opd)
print("Objective = ", value(model.objective))
```

```
Status: Optimal
|       |A      |B      |C      |
|:------|:------|:------|:------|
|W1     |1500.0 |0.0    |0.0    |
|W2     |0.0    |900.0  |800.0  |
Objective = 757700.0
```

Summary

Solve Model

- Call the `solve()` method
- Check the status of the solution
- Print values of decision variables
- Print value of objective function

Let's practice!

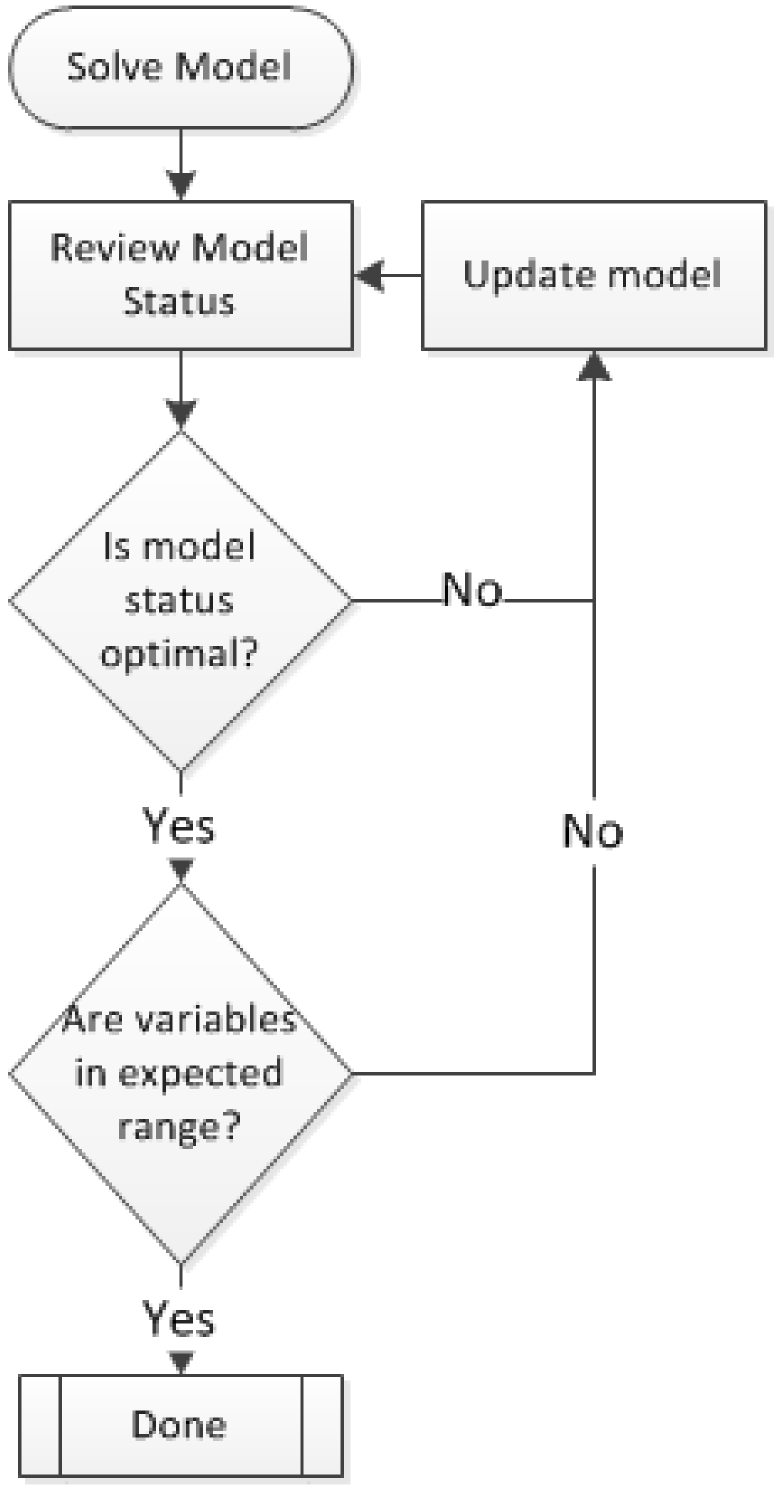
SUPPLY CHAIN ANALYTICS IN PYTHON

datacamp

Sanity checking the solution

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.


Solve Model
Review Model Status
Update model
Is model status optimal?
No
Yes
No
Are variables in expected range?
Yes
Done

Check the model status

- **Infeasible:** There are no feasible solutions.
  - Review the constraints
- **Unbounded:** The object function is not bounded, maximizing or minimizing the objective will tend towards infinity.
  - Review the objective function
- **Undefined:** The optimal solution may exist but may not have been found.
  - Maybe the best available solution
  - Review how you are modeling the problem

Check if results are within expectations

Are the **decision variables** and value of **objective** within expected range?

- Based on knowledge / understanding of problem
- If "Yes", then you have a valid solution
- If "No", then review:
  - Python code
  - Data
  - Write the LP File

Write LP

```
writeLP(filename)
```

- `filename` = The name of the file to be created

Shows:

- Name of problem
- Objective function and if minimizing or maximizing
- Constraints, including constraints on Decision Variables called Bounds
- Decision variables

Code example

```
\* Aggregate Production Planning *\
Minimize
OBJ: 20 prod_('A',_0) + 17 prod_('A',_1)
    + 18 prod_('A',_2) + 15 prod_('B',_0)
    + 16 prod_('B',_1) + 15 prod_('B',_2)
Subject To
_C1: prod_('A',_0) >= 0
_C2: prod_('A',_1) >= 0
_C3: prod_('A',_2) >= 0
_C4: prod_('B',_0) >= 8
_C5: prod_('B',_1) >= 7
_C6: prod_('B',_2) >= 6
```

```
Bounds
0 <= prod_('A',_0)
0 <= prod_('A',_1)
0 <= prod_('A',_2)
0 <= prod_('B',_0)
0 <= prod_('B',_1)
0 <= prod_('B',_2)
Generals
prod_('A',_0)
prod_('A',_1)
prod_('A',_2)
prod_('B',_0)
prod_('B',_1)
prod_('B',_2)
```

Summary

Strategy for Sanity Checking

- Check the model status
- Check decision variables and objective inside expected range
- Use `writeLP()` if needed

Practice time!

SUPPLY CHAIN ANALYTICS IN PYTHON

datacamp

Shadow price sensitivity analysis

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

Define shadow price

Modeling in issues:

- Input for model constraints are often estimates
- Will changes to input change our solution?

Shadow Prices:

- *The change in optimal value of the objective function per unit increase in the right-hand-side for a constraint, given everything else remain unchanged.*

Context - Glass Company - Resource Planning:

Resource	Prod. A	Prod. B	Prod. C
Production hours	6	5	8
WH Capacity sq. ft.	10.5	20	10
Profit $US	$500	$450	$600

Constraints:

- Production Capacity Hours ≤ 60
- Warehouse Capacity ≤ 150 sq. ft.
- Max Production of A ≤ 8

Code example

```python
# Initialize Class, Define Vars., and Objective
model = LpProblem("Max Glass Co. Profits",
                  LpMaximize)
A = LpVariable('A', lowBound=0)
B = LpVariable('B', lowBound=0)
C = LpVariable('C', lowBound=0)
model += 500 * A + 450 * B + 600 * C

# Constraint 1
model += 6 * A + 5 * B + 8 * C <= 60

# Constraint 2
model += 10.5 * A + 20 * B + 10 * C <= 150
```

```python
# Constraint 3
model += A  <= 8

# Solve Model
model.solve()
print("Model Status:
        {}".format(LpStatus[model.status]))
print("Objective = ", value(model.objective))
for v in model.variables():
    print(v.name, "=", v.varValue)
```

Example solution

Solution:

Products	Prod. A	Prod. B	Prod. C
Production Cases	6.667	4	0

Objective value is $5133.33

Review constraints

Decision Variable:

- A through C = Number of cases of respective A through C products

Constraints:

- $6A + 5B + 8C \leq 60$ (*limited production capacity*)
- $10A + 20B + 10C \leq 150$ (*limited warehouse capacity*)
- $A \leq 8$ (*max production of A*)

Print shadow price

Python Code:

```
o = [{'name':name, 'shadow price':c.pi}
     for name, c in model.constraints.items()]
print(pd.DataFrame(o))
```

Shadow prices explained

Output:

```
  name  shadow price
   _C1     78.148148
   _C2      2.962963
   _C3     -0.000000
```

Remember the Constraints:

1. *limited production capacity*
2. *limited warehouse capacity*
3. *max production of A*

Constraint slack

`slack` :

- The amount of a resource that is unused.

Python:

```
o = [{'name':name, 'shadow price':c.pi, 'slack': c.slack}
     for name, c in model.constraints.items()]
print(pd.DataFrame(o))
```

Constraint slack explained

Output:

```
  name   shadow price      slack
   _C1      78.148148 -0.000000
   _C2       2.962963 -0.000000
   _C3      -0.000000  1.333333
```

More About Binding

- `slack` = 0, then ***binding***
- Changing ***binding*** constraint, ***changes*** solution

Remember the Constraints:

1. *limited production capacity*
2. *limited warehouse capacity*
3. *max production of A*

Summary

- How to compute:
  - `shadow prices`
  - constraint `slack`
- Identify Binding Constraints
  - `slack` = 0, then ***binding***
  - `slack` > 0, then ***not-binding***

Try it out!

SUPPLY CHAIN ANALYTICS IN PYTHON

Capacitated plant location - case study P3

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

datacamp

Capacitated plant location model

Modeling

- Production at regional facilities
  - Two plant sizes (low / high)
- Exporting production to other regions
- Production facilities open / close

Expected ranges

What should we expected for values of our decision variables?

Production Quantities:

- High production in regions with low variable production and shipping costs
- Maxed production in regions that also have relatively low fixed production costs

Production Plant Open Or Closed:

- High capacity production plant in regions with high demand
- High capacity production plant in regions with relatively low fixed costs

Sensitivity analysis of constraints

Total Production = Total Demand:

- `shadow prices` = Represent changes in total cost per increase in demand for a region
- `slack` = Should be zero

Total Production ≤ Total Production Capacity:

- `shadow prices` = Represent changes in total costs per increase in production capacity
- `slack` = Regions which have excess production capacity

```python
from pulp import *
import pandas as pd

# Initialize Class
model =
    LpProblem("Capacitated Plant Location Model",
              LpMinimize)

# Define Decision Variables
loc = ['A', 'B', 'C', 'D', 'E']
size = ['Low_Cap','High_Cap']
x = LpVariable.dicts(
               "production_",
               [(i,j) for i in loc for j in loc],
                lowBound=0, upBound=None,
                cat='Continuous')
```

```python
y = LpVariable.dicts(
               "plant_",
               [(i,s) for s in size for i in loc],
                cat='Binary')
# Define Objective Function
model +=
  (lpSum([fix_cost.loc[i,s]*y[(i,s)]
          for s in size for i in loc])
 + lpSum([var_cost.loc[i,j]*x[(i,j)]
          for i in loc for j in loc]))

# Define the Constraints
for j in loc: model +=
  lpSum([x[(i, j)]
        for i in loc]) == demand.loc[j,'Dmd']
for i in loc: model +=
  lpSum([x[(i, j)] for j in loc]) <= lpSum(
             [cap.loc[i,s]*y[(i,s)]for s in size])
```

```python
# Solve
model.solve()
# Print Decision Variables and Objective Value
print(LpStatus[model.status])
o = [{'prod':"{} to {}".format(i,j), 'quant':x[(i,j)].varValue}
     for i in loc for j in loc]
print(pd.DataFrame(o))
o = [{'loc':i, 'lc':y[(i,size[0])].varValue, 'hc':y[(i,size[1])].varValue}
     for i in loc]
print(pd.DataFrame(o))
print("Objective = ", value(model.objective))
# Print Shadow Price and Slack
o = [{'name':name, 'shadow price':c.pi, 'slack': c.slack}
     for name, c in model.constraints.items()]
print(pd.DataFrame(o))
```

Business questions

Likely Questions:

- What is the expected cost of this supply chain network model?
- If demand increases in a region how much profit is needed to cover the costs of production and shipping to that region?
- Which regions still have production capacity for future demand increase?

Summary

Reviewed:

- Expected ranges for decision variables
- Interpreted the output of sensitivity analysis (`shadow prices` and `slack`)
- Code to solve and output results
- Likely business related question

Great work! Your turn

SUPPLY CHAIN ANALYTICS IN PYTHON

datacamp

Simulation testing solution

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

datacamp

Caution

- Problems that take a long time to solve should not be used with LP or IP

Overall concept

General Concept:

- Add random noise to key inputs you choose
- Solve the model repeatedly
- Observe the distribution

Why we might try

Why:

- Inputs are often estimates. There is a risk that they are inaccurate.
- Earlier Sensitivity Analysis only looked at changing one input at a time.

Context

Context - Glass Company - Resource Planning:

Resource	Prod. A	Prod. B	Prod. C
Profit $US	$500	$450	$600

Constraints:

- There are demand, production capacity, and warehouse Capacity constraints

Risks:

- Estimates of profits may be inaccurate

```python
# Initialize Class, & Define Variables
model = LpProblem("Max Glass Co. Profits", LpMaximize)
A = LpVariable('A', lowBound=0)
B = LpVariable('B', lowBound=0)
C = LpVariable('C', lowBound=0)

# Define Objective Function
model += 500 * A + 450 * B + 600 * C

# Define Constraints & Solve
model += 6 * A + 5 * B + 8 * C <= 60
model += 10.5 * A + 20 * B + 10 * C <= 150
model += A  <= 8
model.solve()
```

Code example - step 2

```
a, b, c = normalvariate(0,25),
          normalvariate(0,25),
          normalvariate(0,25)
```

```
# Define Objective Function
model += (500+a)*A + (450+b)*B + (600+c)*C
```

```
# Initialize Class, & Define Variables
model = LpProblem("Max Glass Co. Profits",
                   LpMaximize)
```

```
A = LpVariable('A', lowBound=0)
B = LpVariable('B', lowBound=0)
C = LpVariable('C', lowBound=0)
a, b, c = normalvariate(0,25),
          normalvariate(0,25),
          normalvariate(0,25)
# Define Objective Function
model += (500+a)*A + (450+b)*B + (600+c)*C

# Define Constraints & Solve
model += 6 * A + 5 * B + 8 * C <= 60
model += 10.5 * A + 20 * B + 10 * C <= 150
model += A  <= 8
model.solve()
```

```python
def run_pulp_model():
    # Initialize Class
    model = LpProblem("Max Glass Co. Profits", LpMaximize)
    A = LpVariable('A', lowBound=0)
    B = LpVariable('B', lowBound=0)
    C = LpVariable('C', lowBound=0)
    a, b, c = normalvariate(0,25), normalvariate(0,25), normalvariate(0,25)

    # Define Objective Function
    model += (500+a)*A + (450+b)*B + (600+c)*C

    # Define Constraints & Solve
    model += 6 * A + 5 * B + 8 * C <= 60
    model += 10.5 * A + 20 * B + 10 * C <= 150
    model += A  <= 8
    model.solve()
    o = {'A':A.varValue, 'B':B.varValue, 'C':C.varValue, 'Obj':value(model.objective)}
    return(o)
```

Code example - step 4

```
def run_pulp_model():
    # Initialize Class
    model = LpProblem("Max Glass Co. Profits",
                      LpMaximize)
    A = LpVariable('A', lowBound=0)
    B = LpVariable('B', lowBound=0)
    C = LpVariable('C', lowBound=0)
    a, b, c = normalvariate(0,25),
              normalvariate(0,25),
              normalvariate(0,25)

    # Define Objective Function
    model += (500+a)*A + (450+b)*B +
             (600+c)*C
```

```
    # Define Constraints & Solve
    model += 6 * A + 5 * B + 8 * C <= 60
    model += 10.5 * A + 20 * B + 10 * C <= 150
    model += A  <= 8
    model.solve()
    o = {'A':A.varValue, 'B':B.varValue,
         'C':C.varValue,
         'Obj':value(model.objective)}
    return(o)
```

```
for i in range(100):
    output.append(run_pulp_model())
df = pd.DataFrame(output)
```

Code example - step 5

```
print(df['A'].value_counts())
print(df['B'].value_counts())
print(df['C'].value_counts())
```

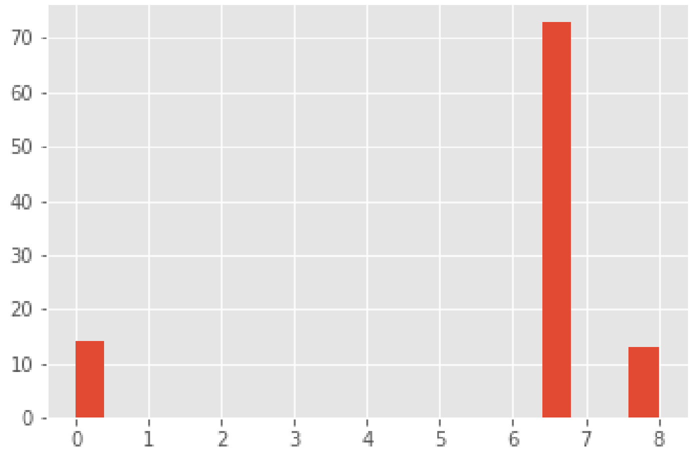
Output: (results may be different)

```
6.666667    73
0.000000    14
8.000000    13
Name: A, dtype: int64
```

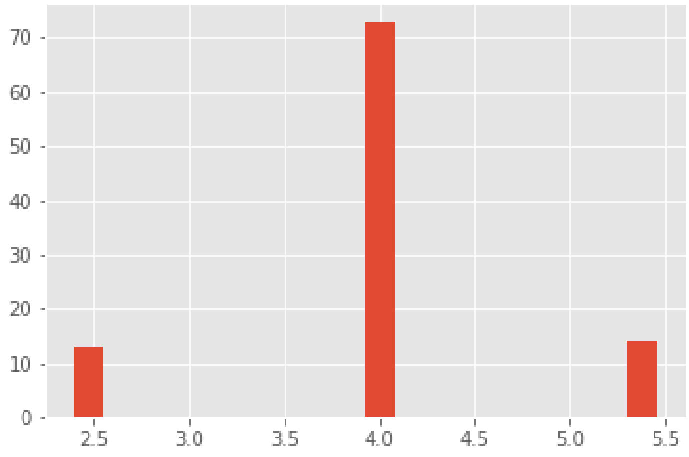
```
4.000000    73
5.454546    14
2.400000    13
Name: B, dtype: int64
0.000000    86
4.090909    14
```

Visualize as histogram

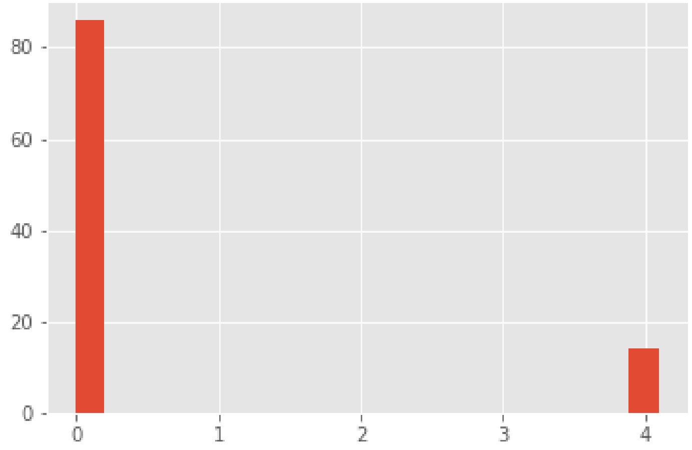
Product A:



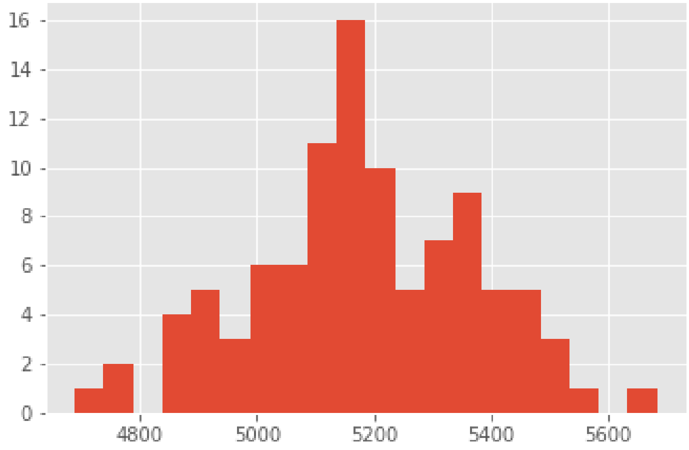
Product B:



Product C:



Objective Values:



Summary

- Should not be used on problems that take a long time to solve
- Benefits
  - View how optimal results change as model inputs change
- Steps
  1. Start with standard PuLP model code
  2. Add noise to key inputs using Python's `normalvariate`
  3. Wrap PuLP model code in a function that returns the model's output
  4. Create loop to call newly created function and store results in DataFrame
  5. Visualize results DataFrame

Try it out!

SUPPLY CHAIN ANALYTICS IN PYTHON

Capacitated plant location - case study P4

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr., Ingredion

datacamp

Simulation vs. sensitivity analysis

With Sensitivity Analysis:

- Observe how changes in demand and costs affect production:
  - Where should production be added?
  - Does production move to a different region.
  - Which regions have stable production quantities?
- Observe multiple changes at once vs. one at a time with sensitivity analysis

Simulation modeling

We can apply simulation testing to our Capacitated Plant Location Model

Possible inputs for adding noise

- **Demand**
- **Variable costs**
- Fixed costs
- Capacity

```python
# Initialize Class
model = LpProblem(
            "Capacitated Plant Location Model",
             LpMinimize)

# Define Decision Variables
loc = ['A', 'B', 'C', 'D', 'E']
size = ['Low_Cap','High_Cap']
x = LpVariable.dicts(
       "production_",
       [(i,j) for i in loc for j in loc],
       lowBound=0, upBound=None, cat='Continuous')
y = LpVariable.dicts(
      "plant_", [(i,s)for s in size for i in loc],
       cat='Binary')
```

```python
# Define Objective Function
model +=(lpSum([fix_cost.loc[i,s]*y[(i,s)]
               for s in size for i in loc])
       + lpSum([var_cost.loc[i,j]*x[(i,j)]
                for i in loc for j in loc]))


# Define the Constraints
for j in loc: model +=
  lpSum([x[(i, j)] for i in loc]) == demand.loc[
                                         j,'Dmd']
for i in loc: model +=
  lpSum([x[(i, j)] for j in loc]) <= lpSum(
                             [cap.loc[i,s]*y[(i,s)]
                              for s in size])
# Solve
model.solve()
print(LpStatus[model.status])
```

Objective:

```
model += (lpSum([fix_cost.loc[i,s]*y[(i,s)] for s in size for i in loc])
          + lpSum([(var_cost.loc[i,j] + normalvariate(0.5, 0.5))*x[(i,j)]
                   for i in loc for j in loc]))
```

Total Demand:

```
for j in loc:
    rd = normalvariate(0, demand.loc[j,'Dmd']*.05)
    model += lpSum([x[(i,j)] for i in loc]) == (demand.loc[j,'Dmd']+rd)
```

Code example - step 3

```
def run_pulp_model(fix_cost, var_cost, demand,
                   cap):
    # Initialize Class
    model = LpProblem(
              "Capacitated Plant Location Model",
               LpMinimize)

    # Define Decision Variables
    loc = ['A', 'B', 'C', 'D', 'E']
    size = ['Low_Cap','High_Cap']
    x = LpVariable.dicts(
                "production_",
                [(i,j) for i in loc for j in loc],
                lowBound=0, upBound=None,
                cat='Continuous')
```

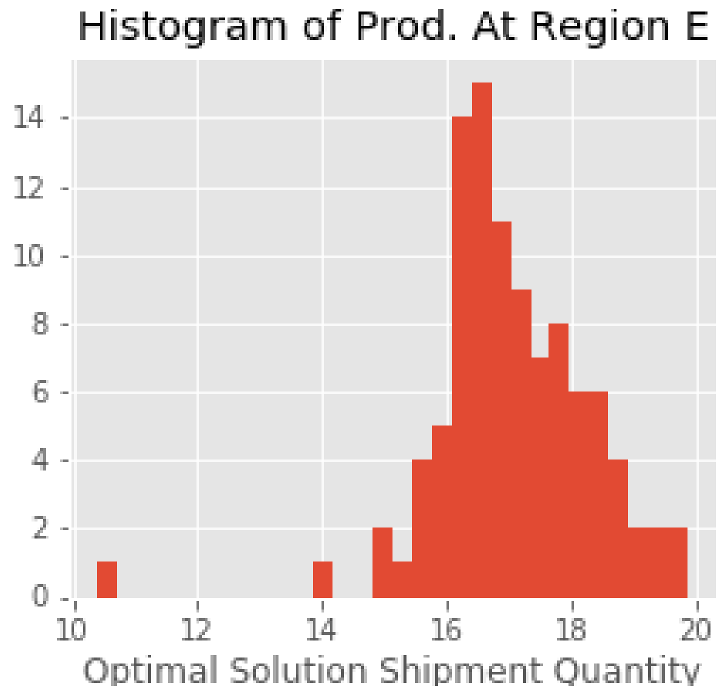
```
    y = LpVariable.dicts(
                "plant_",
                [(i,s) for s in size for i in loc],
                 cat='Binary')
    # Define the Constraints
    for j in loc: rd = normalvariate(
                        0, demand.loc[j,'Dmd']*.05)
        model += lpSum(
         [x[(i,j)] for i in loc]) == (
                           demand.loc[j,'Dmd']+rd)
    for i in loc: model +=
      lpSum([x[(i,j)] for j in loc]) \
        <= lpSum([cap.loc[i,s]*y[(i,s)]
            for s in size])
```

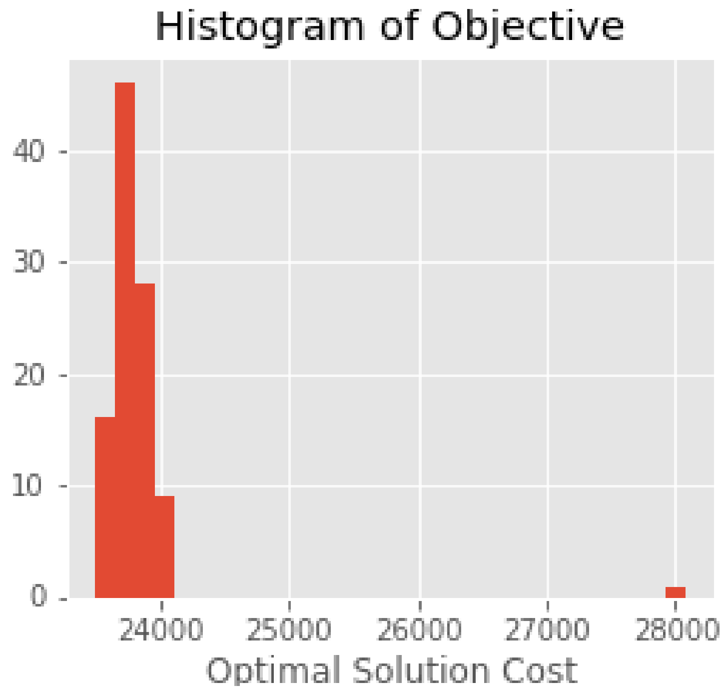
```python
    # Solve
    model.solve()
    o = {}
    for i in loc:
        o[i] = value(lpSum([x[(i, j)] for j in loc]))
    o['Obj'] = value(model.objective)
    return(o)
```

```python
for i in range(100):
    output.append(run_pulp_model(fix_cost, var_cost, demand, cap))
df = pd.DataFrame(output)
```

Results

```
import matplotlib.pyplot as plt
plt.title('Histogram of Prod. At Region E')
plt.hist(df['E'])
plt.show()
```





Summary

Capacitated Plant Model

- Simulation vs. sensitivity analysis
- Stepped through code example

Try it out!

SUPPLY CHAIN ANALYTICS IN PYTHON

Final summary

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

Summary

- Reviewed what is Linear Programing (LP)
- Reviewed PuLP and how it can be used with LP
- Solving large scale models
  - `LpSum()`
  - `LpVariable.dicts()`
- Logical constraints
- Common constraint mistakes
- Solving PuLP model
  - printing decision variables, and objective

Summary

- Sanity checking solution
- Sensitivity Analysis
  - Shadow Prices
  - Slack
- Simulation Testing
- Capacitated Plant Location model - Case Study

Congratulations!

Additional resources

For more on PuLP check out these additional resources:

- https://www.coin-or.org/PuLP/
- https://www.coin-or.org/
- PuLP GitHub: https://github.com/coin-or/pulp
- Google group: https://groups.google.com/forum/#!forum/pulp-or-discuss

Additional resources

For books related to the subject, check out these:

- Bradley, Stephen P., et al. *Applied Mathematical Programming*. Addison-Wesley, 1977.
- Chopra, Sunil, and Meindl, Peter. *Supply Chain Management: Strategy, Planning, and Operations.* Pearson Prentice-Hall, 2007.

Thank you!

SUPPLY CHAIN ANALYTICS IN PYTHON