Mathematical Programming in Python Using DOcplex

Hilal YILMAZ · Follow

5 min read · Jun 27, 2022

Listen Share

Mathematical programming is the use of mathematical models for decision problems. It is widely used in planning production, transportation, and scheduling. The best part about mathematical programming is that it always guarantees the optimal solution.

Here, we will model a simple problem in python using DOcplex, a python modeling library for optimization. After reading this article, you will learn:

- How to mathematically model a problem
- How to model a problem in python
- How to find an optimal solution with DOcplex

A Warehouse Location Problem

A company is considering opening warehouses at specific locations. The company manager wants to decide where to open warehouses, and how many products should be transported from which warehouse to which customer in order to meet the demands.

The alternative warehouse locations, their storage capacities (orange cells), the unit transportation costs to the customers (grey cells), the customer demands (green cells), and the cost of opening a warehouse (blue cells) are given in Fig. 1:

	Demand Area 1	Demand Area 2	Demand Area 3	Demand Area 4	Capacity	Opening cost
Warehouse 1	4	5	4	3	140	7500
Warehouse 2	1	2	3	1	150	6200
Warehouse 3	2	2	2	3	170	7100
Demand	50	70	40	50		

Fig 1. Problem data

Problem Data and Parameters

According to the given data, we have two sets: warehouses and demand areas, and four parameters are given in Fig. 2:

W: Set of warehouses

D: Set of demand areas

f_i: The cost of opening a warehouse at location i

s_i: Storage capacity of warehouse i

d_j: Demand of demand area j

c_{ij}: Unit transportation cost from warehouse i to demand area j

Fig 2. Notations of the problem data and parameters

Decision Variables

We have two decisions to make. The notations of the decision variables are shown in Fig. 3:

1. The decision of the number of products to transfer from each warehouse to each customer (Integer variable)
2. The decision of opening a warehouse (Binary variable)

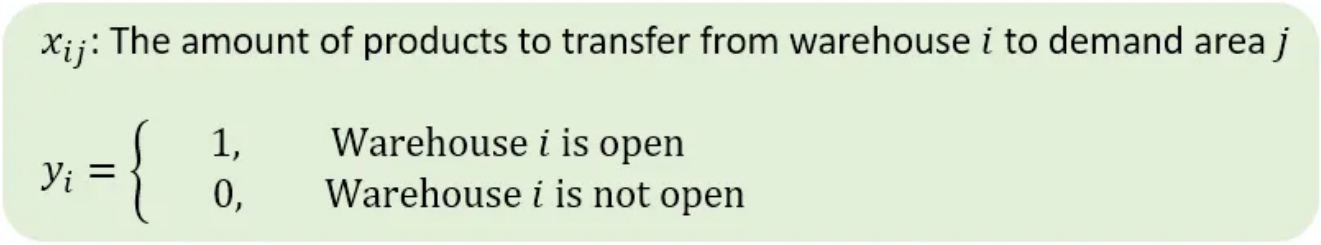


Fig 3. The decision variables

Modeling in Python

This problem has 3 alternative warehouse locations and 4 demand areas. Let's start by defining these datasets in python:

```
#Define Data
warehouses=list(range(3))
demand_areas=list(range(4))
```



```
#storage capacity
s={warehouses[0]:140, warehouses[1]:150, warehouses[2]:170}

#opening cost
f={warehouses[0]:7500, warehouses[1]:6200,warehouses[2]:7100}

#demand
d={demand_areas[0]:50, demand_areas[1]:70, demand_areas[2]:40,
demand_areas[3]:50}

#unit transportation cost

c={(warehouses[0],demand_areas[0]):4,
   (warehouses[0],demand_areas[1]):5,
   (warehouses[0],demand_areas[2]):4,
   (warehouses[0],demand_areas[3]):3,
   (warehouses[1],demand_areas[0]):1,
   (warehouses[1],demand_areas[1]):2,
   (warehouses[1],demand_areas[2]):3,
   (warehouses[1],demand_areas[3]):1,
   (warehouses[2],demand_areas[0]):2,
   (warehouses[2],demand_areas[1]):2,
   (warehouses[2],demand_areas[2]):2,
   (warehouses[2],demand_areas[3]):3}
```

Before defining the decision variables in python, we need to include the class “Model” from the DOcplex library and define a new model object.

```
#Build Model
from docplex.mp.model import Model
mdl = Model(name="warehouse_location")
```

When adding the variables, we need to specify the variable type, whether it takes binary values (0,1), integer values, or continuous values. In this problem, we have “y” as the binary variable and “x” as the integer variable. We can also specify the names of the variable.

```
1  #Variables
2  y = {i: mdl.binary_var(name="y%d" % (i+1)) for i in warehouses}
3  x = {(i,j): mdl.integer_var(name="x%d%d" % (i+1,j+1)) for i in warehouses for j in demar
```

variables.py hosted with ❤ by GitHub view raw

Objective Function

The warehouse location problem has two types of costs: the cost of opening a warehouse and the cost of transporting products from warehouses to demand areas. The general form and the open are given in Fig. 4.

Minimization of total cost	Minimization of total cost
$\sum_{i=1}^{W} f_i y_i + \sum_{i=1}^{W}\sum_{j=1}^{D} c_{ij} x_{ij}$	$7500y_1 + 6200y_2 + 7100y_3 + 4x_{11} + 5x_{12} + 4x_{13} + 3x_{14} + 1x_{21} + 2x_{22} + 3x_{23} + 1x_{24} + 2x_{31} + 2x_{32} + 2x_{33} + 3x_{34}$

Fig 4. The objective function of the warehouse location problem

Our aim is to minimize the sum of these costs. With Docplex library, we define the objective function according to the objective, whether it is minimization or maximization.

```
#Objective function
mdl.minimize( mdl.sum(f[i]*y[i] for i in warehouses) +
mdl.sum(c[i,j]*x[i,j] for i in warehouses for j in demand_areas))
```

Constraints

The cost should be minimized regarding the **demand satisfaction** and **capacity restrictions.**

Demand satisfaction means that the number of total products transported to a demand area, should not be less than the demand. The mathematical form of this constraint is given in Fig. 5.

Fig 5. Demand satisfaction constraint

To include constraints in our model in python, we use the “add_constraint()” function. The demand satisfaction constraint is added for each demand area. We can easily write this constraint in a for-loop. We can also print the constraints if we want to observe them.

```
#Demand satisfaction
for j in demand_areas:
    demand_satisfaction = mdl.add_constraint(mdl.sum(x[i, j] for i in warehouses) >= d[j
    print(demand_satisfaction)
```

demand_satisfaction.py hosted with ❤ by GitHub view raw

The last constraint is the capacity restriction constraint. This means that the number of total products transferred from a warehouse should not be greater than the capacity of the warehouse. This constraint can be applied only if the warehouse is opened. However, if the warehouse is not going to be open, no product can be transported from the warehouse.

This is how we mathematically define the capacity restriction (Fig. 6):

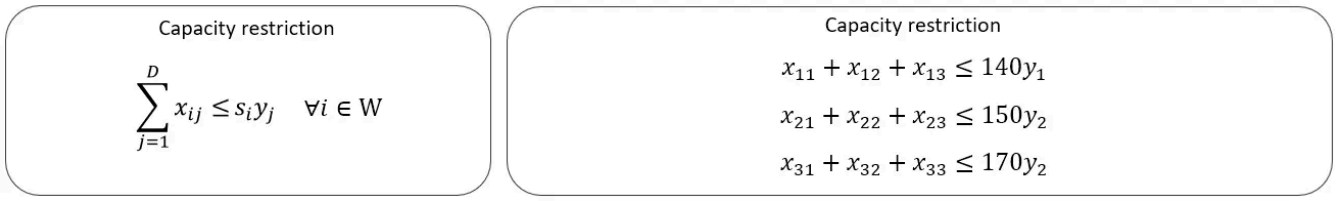


Fig 6. Capacity restriction constraint

And this is how we define it in python:

```
#Capacity restriction
for i in warehouses:
    capacity_restriction = mdl.add_constraint(mdl.sum(x[i, j] for j in demand_areas)<=s[
    print(capacity_restriction)
```

capacity_restriction.py hosted with ❤ by GitHub view raw

Solving the model

We're almost done :) Now, all we need to do is solve the model. To see the result we use print_solution().

```
if mdl.solve():
    mdl.print_solution()
```

Optimal Solution

After running the code the result we obtain is this:

```
x11+x21+x31 >= 50
x12+x22+x32 >= 70
x13+x23+x33 >= 40
x14+x24+x34 >= 50
x11+x12+x13+x14 <= 140y1
x21+x22+x23+x24 <= 150y2
x31+x32+x33+x34 <= 170y3
-----------------------------
objective: 13620
  y2=1
  y3=1
  x21=50
  x22=50
  x24=50
  x32=20
  x33=40
```

According to the optimal solution, the minimum total cost is 13620 currency units if the company manager opens two warehouses at location 2 and location 3. The amount of product transportation from warehouse 2 to demand areas 1, 2, and 4 is 50. The amount of product transportation from warehouse 3 to demand area 2 is 20, and to demand area 3 is 40.

The complete code is given below:

```
#Define Data
warehouses=list(range(3))
demand_areas=list(range(4))

#unit transportation cost
c={(warehouses[0],demand_areas[0]):4,
   (warehouses[0],demand_areas[1]):5,
   (warehouses[0],demand_areas[2]):4,
   (warehouses[0],demand_areas[3]):3,
   (warehouses[1],demand_areas[0]):1,
   (warehouses[1],demand_areas[1]):2,
   (warehouses[1],demand_areas[2]):3,
   (warehouses[1],demand_areas[3]):1,
   (warehouses[2],demand_areas[0]):2,
   (warehouses[2],demand_areas[1]):2,
   (warehouses[2],demand_areas[2]):2,
   (warehouses[2],demand_areas[3]):3}

s={warehouses[0]:140, warehouses[1]:150, warehouses[2]:170} #storage capacity
f={warehouses[0]:7500, warehouses[1]:6200,warehouses[2]:7100} #opening cost
d={demand_areas[0]:50, demand_areas[1]:70, demand_areas[2]:40, demand_areas[3]:50} #dem

#Build Model
from docplex.mp.model import Model
mdl = Model(name="warehouse_location")
y = {i: mdl.binary_var(name="y%d" % (i+1)) for i in warehouses}
x = {(i,j): mdl.integer_var(name="x%d%d" % (i+1,j+1)) for i in warehouses for j in dema

#Objective function
mdl.minimize( mdl.sum(f[i]*y[i] for i in warehouses) + mdl.sum(c[i,j]*x[i,j] for i in w

#Demand satisfaction
for j in demand_areas:
    demand_satisfaction = mdl.add_constraint(mdl.sum(x[i, j] for i in warehouses) >= d[
    print(demand_satisfaction)

#Capacity restriction
for i in warehouses:
    capacity_restriction = mdl.add_constraint(mdl.sum(x[i, j] for j in demand_areas)<=s
    print(capacity_restriction)

if mdl.solve():
    print("\n-----------------------------")
    mdl.print_solution()
```

Cplex | Mathematical Modeling | Python | Operations Research | Data Science

Follow

Written by Hilal YILMAZ

4 Followers

Industrial Engineering PhD www.linkedin.com/in/hilal-yilmaz0861

Recommended from Medium

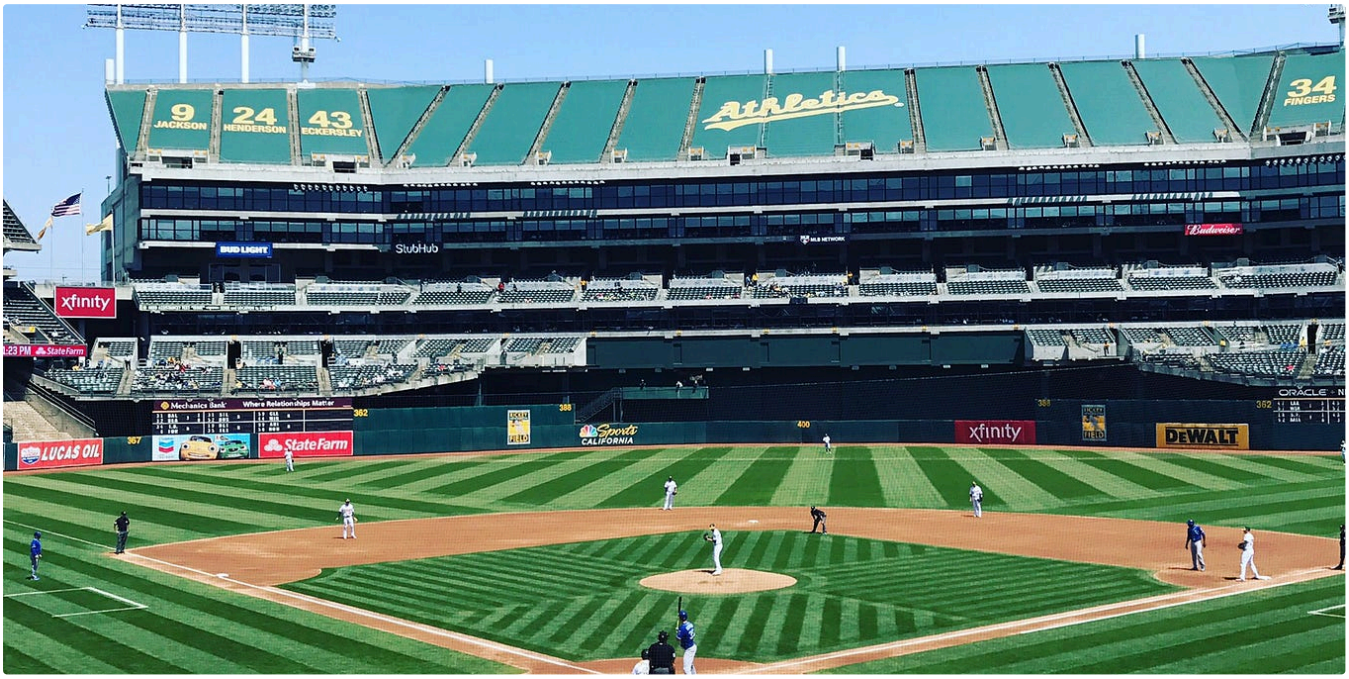
Techmaniac

Creating a Single Executable from Multiple Python Files

Python is a versatile programming language known for its readability and ease of use. While Python code is typically executed by an...

2 min read · Nov 21, 2023

63



Giovanni Malloy in Towards Data Science

Decision Analysis and Trees in Python—The Case of the Oakland A's

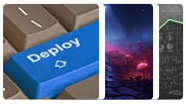
Using decision trees in Python to extract insight into the A's decision to move to Las Vegas

· 17 min read · May 24, 2023

258 3

Lists

Coding & Development
11 stories · 559 saves



Predictive Modeling w/ Python
20 stories · 1091 saves

Practical Guides to Machine Learning
10 stories · 1310 saves

ChatGPT

21 stories · 571 saves

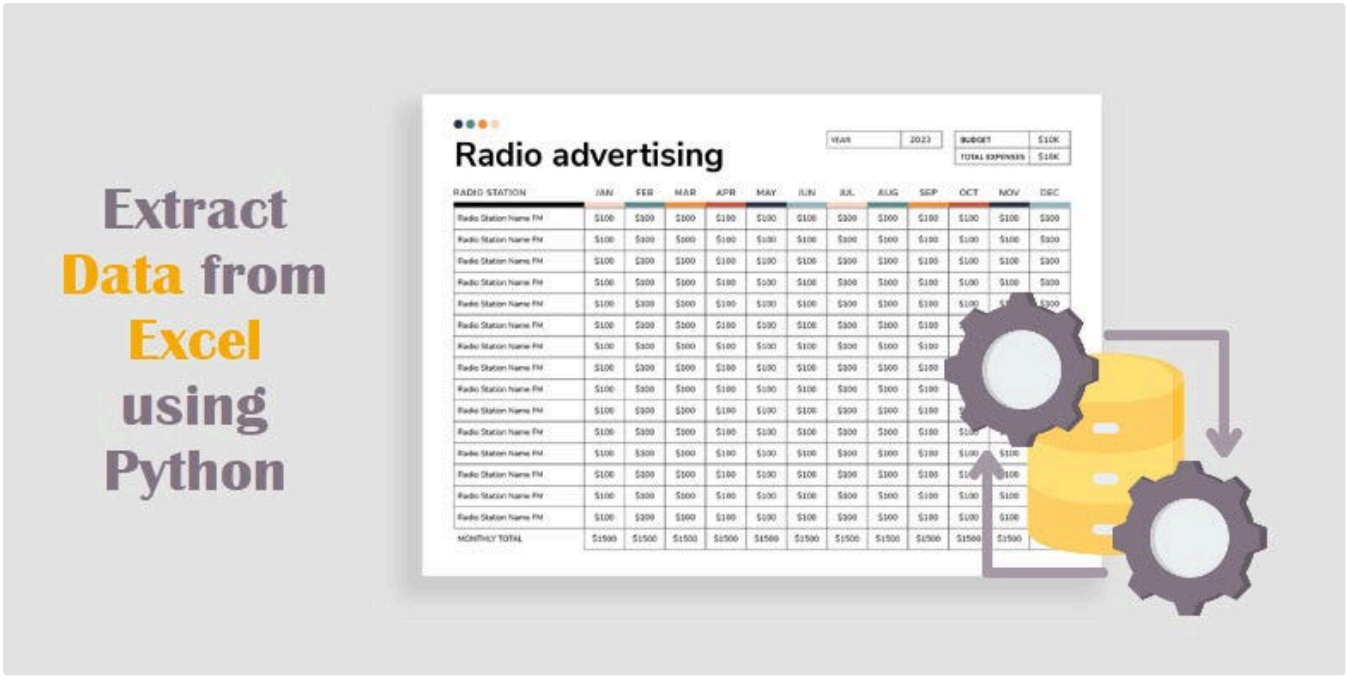
Jason Roell

Ultimate Python Cheat Sheet: Practical Python For Everyday Tasks

This Cheat Sheet was born out of necessity. Recently, I was tasked with diving into a new Python project after some time away from the...

33 min read · Jan 29, 2024

3.7K 36



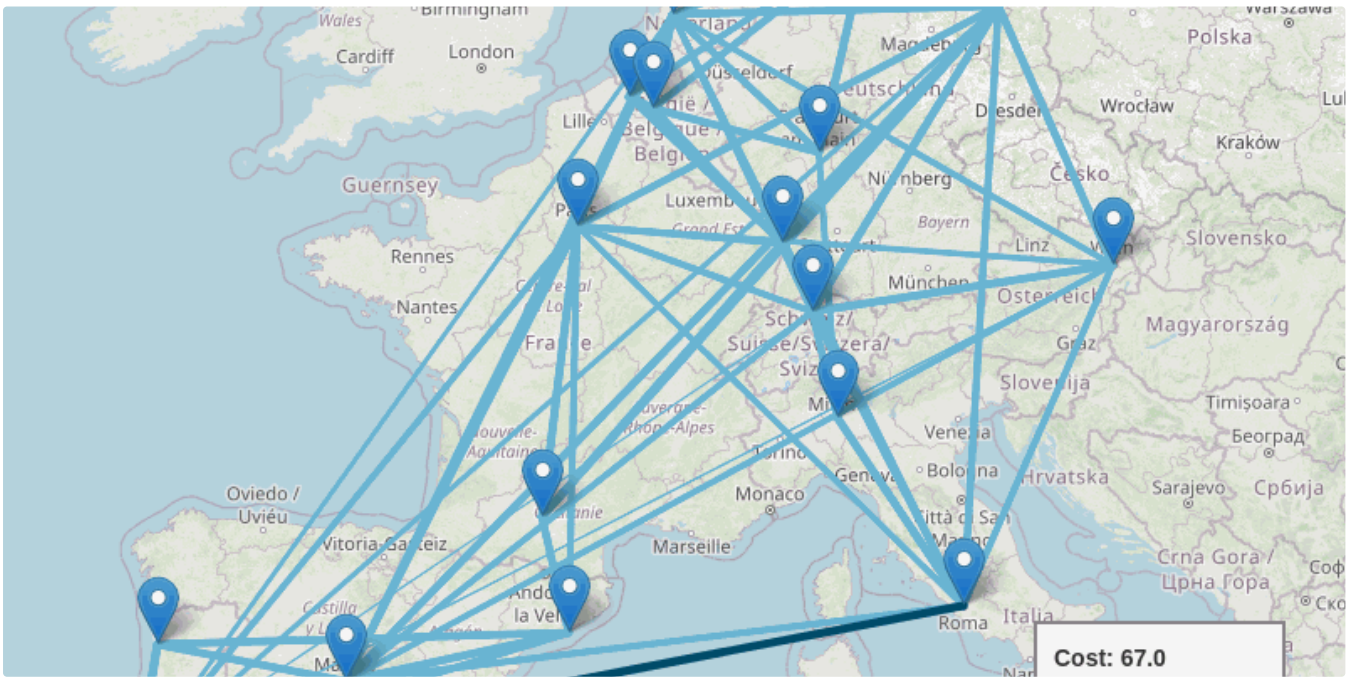
Alexander Stock in Python in Plain English

Extracting Data from Excel with Python: 6 Code Examples

Extracting data from Excel programmatically is a valuable approach that enables automation and efficiency in data retrieval. By leveraging...

6 min read · Mar 12, 2024

9



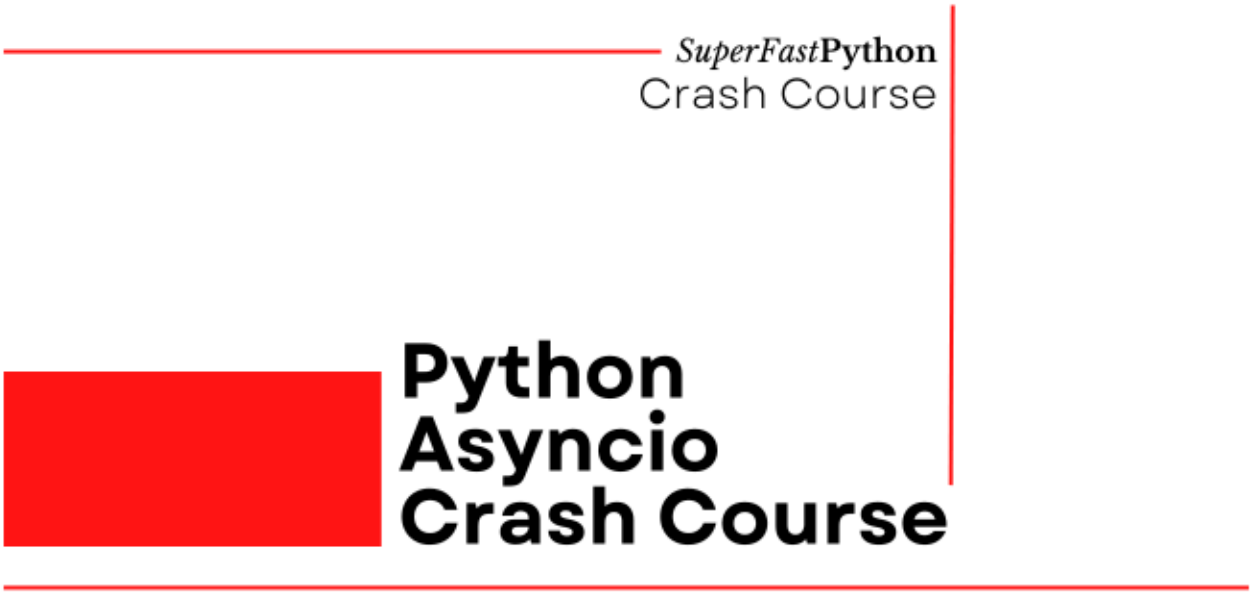
Cristina Fernandes in Data And Beyond

How to solve a Routing Problem with a Genetic Algorithm: A Practical Guide

In the fast-paced world of logistics and transportation, the efficiency of routing from one city to another is a cornerstone of economic...

22 min read · Nov 29, 2023

115 1



SFP Super Fast Python

Python Asyncio: 7-Day Crash-Course

Python Asyncio allows us to use asynchronous programming with coroutine-based concurrency in Python.

13 min read · Nov 19, 2023

73 1

See more recommendations