ANALYTICS TX, LLC

Linear Programming Using Python – Basic Example

Dr. Kruti Lehenbauer

4/14/2023

Example: PAR Insights, Inc. Profit Maximization using Python

In an effort to learn (and share it) Python Programming to perform various Data Analytics operations instead of using the traditional Excel and STATA methods that I have used for over 2 decades, here is the first installment of solving a simple linear programming profit maximization problem in Python. This example was created for the book, *Introduction to Management Science: A Stepwise Approach to Basic Models in a Management Toolkit* (referred to as "the book" for the remainder of this file) and it is available in Kindle and Paperback format on Amazon from here.

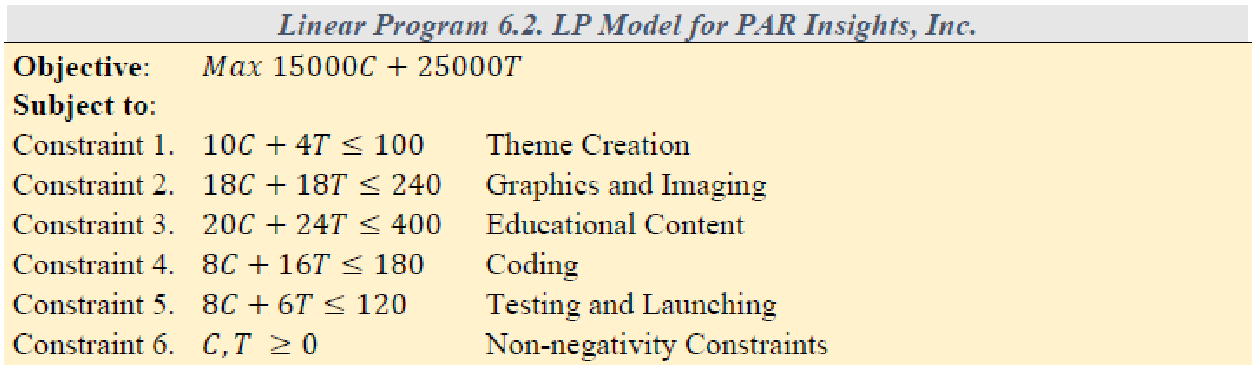
Chapter 6 of the book goes into detailed explanations of what Linear Programming is and how it can be used in business applications to optimize outcomes, so I am not delving too deeply into the background or the theory behind LP in this document. The book also covers step by step methods to solve LP Models using Graphical Methods and Excel Solver, so please feel free to refer to that if you need the stepwise approach.

In this example, we will be focusing on recreating and solving the Example 6.1 titled "PAR Insights, Inc LP Maximization Model" from page 139-143 of the book.

Start by importing pandas, numpy, matplotlib.pyplot, and tabulate in the Python file. The most important import for solving Linear Programming problems in Python is PuLP. You might need to install these packages before you can import them to the workspace. Note that I have skipped some output information in the code below for the sake of brevity.

```python
import pandas as pd
import pulp as p
import numpy as np
import matplotlib.pyplot as plt
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)
from tabulate import tabulate
```

The Linear Program that we need to solve is represented by the following LP Model (see pg. 142 of the book):



Linear Program 6.2. LP Model for PAR Insights, Inc.

Objective:	$Max\ 15000C + 25000T$	
Subject to:		
Constraint 1.	$10C + 4T \leq 100$	Theme Creation
Constraint 2.	$18C + 18T \leq 240$	Graphics and Imaging
Constraint 3.	$20C + 24T \leq 400$	Educational Content
Constraint 4.	$8C + 16T \leq 180$	Coding
Constraint 5.	$8C + 6T \leq 120$	Testing and Launching
Constraint 6.	$C, T \geq 0$	Non-negativity Constraints

Now, we can try to create this in Python in the following stepwise manner:

First, we put in some text to describe what the Problem is and set up the type of model that we want to solve in PuLp by "initializing" the model (PARM in our example) as shown below:

```
# PAR Insights Example 6_1
# The Problem: PAR Insights is trying to expand its business into the educational apps market and can create two types of products with which they can enter the market.
# The management is trying to identify how many apps of each type should they launch in the next four months.

# Initialize the LP
PARM = p.LpProblem("PAR_Insights_Profit_Max", p.LpMaximize)
```

In the Linear Program described, there are two main variables that need to be estimated: the number of Children's Apps (C) and the number of Teenager's Apps (T). The constraints come from the number of hours available within each department of the company. So, let us define the Decision Variables in Python, next.

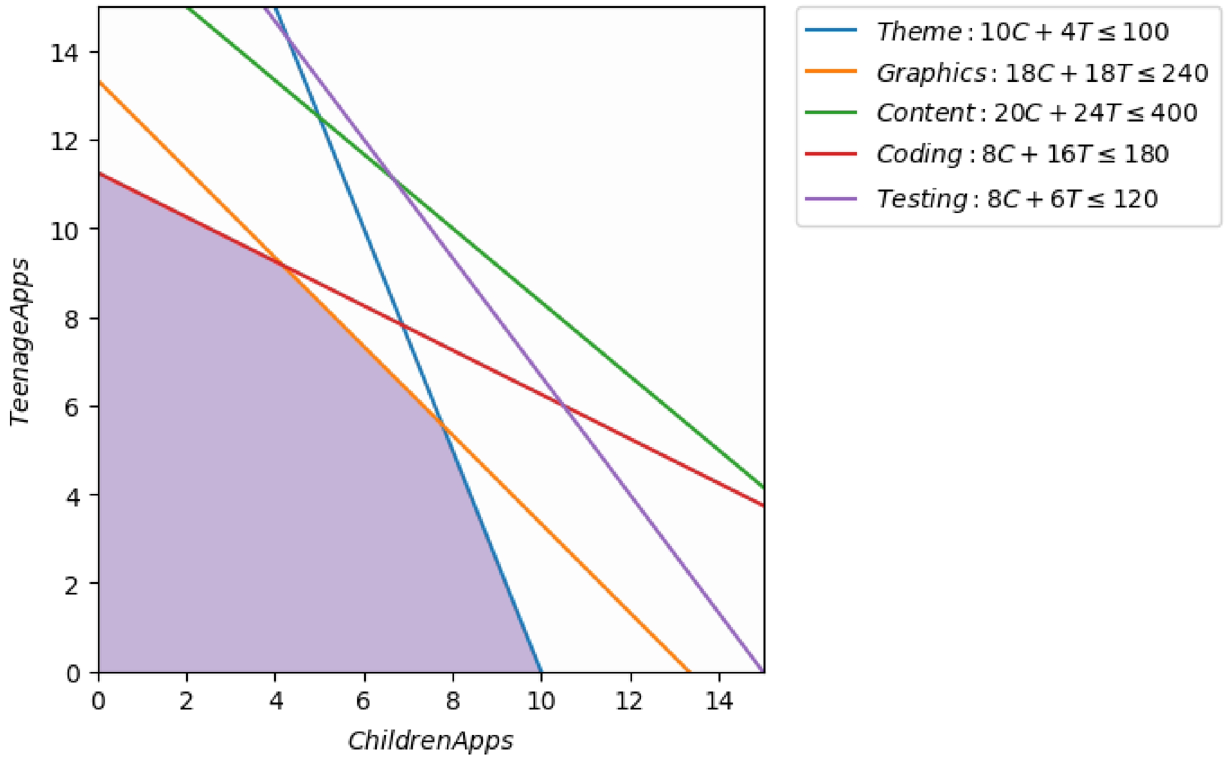
```
# Decision Variables
C = p.LpVariable(name = "Children Apps", lowBound = 0)
T = p.LpVariable(name = "Teenager Apps", lowBound = 0)
```

The p.LpVariable assigns the variable names and their descriptions to the LP Model. Next, we will define the Objective Function (the function that we need to maximize) and the Constraints that need to be added to our model (PARM) as shown below.

```
# Objective Function
PARM += 15000*C + 25000*T

# Constraints
PARM += (10*C + 4*T <= 100, "Theme")
PARM += (18*C + 18*T <= 240, "Graphics & Imaging")
PARM += (20*C + 24*T <= 400, "Educational Content")
PARM += (8*C + 16*T <= 180, "Coding")
PARM += (8*C + 6*T <= 120, "Testing & Launching")
```

Before proceeding to solving this model, it might help to see how these constraints can be visualized in a graph and to identify the region of feasibility for this problem. Note that since we have ONLY two variables in this model, we are able to visualize them in a 2-D graph. If we have larger number of variables, that would not be feasible. This graph has been created by using matplotlib.pyplot tools. The code for this is shown in the Appendix 1 below at the end of the PAR Insights Example.

Chart 1. Feasibility Region for PAR Insights, Inc LP Problem



One can see that the three constraints that define the region of feasibility are the Theme, Graphics, and Coding Department constraints. The Educational Content and Testing constraints are therefore not impacting the solution are likely to have a lot of "slack" or unused resources within them. Now, we can solve the problem by using the "solve" function. There are many ways to see the relevant output. I have defined a special function for output called "pretty_output()" and the code can be found in Appendix 2 below.

```
# Solving the model:
PARsol1 = PARM.solve()
pretty_output(PARM)
```

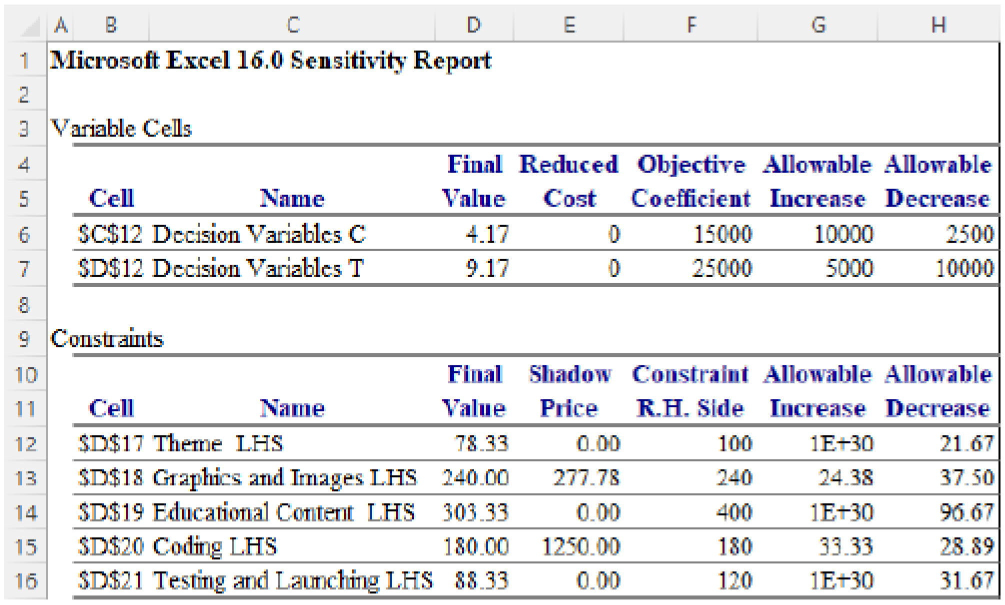
This will yield the following output below the cell (if you are using ipynb Jupyter notebook).

```
PAR_Insights_Profit_Max:
MAXIMIZE
15000*Children_Apps + 25000*Teenager_Apps + 0
SUBJECT TO
Theme: 10 Children_Apps + 4 Teenager_Apps <= 100
Graphics_&_Imaging: 18 Children_Apps + 18 Teenager_Apps <= 240
Educational_Content: 20 Children_Apps + 24 Teenager_Apps <= 400
Coding: 8 Children_Apps + 16 Teenager_Apps <= 180
Testing_&_Launching: 8 Children_Apps + 6 Teenager_Apps <= 120

VARIABLES
Children_Apps Continuous
Teenager_Apps Continuous

Model status: 1, Optimal
```

```
Value of Objective function: 291,666.67
Variable Cells Table:
+---------------+-----------------+----------------+-------------------------+
|     Name      |  Optimal Value  |  Reduced Cost  |  Objective Coefficient  |
+===============+=================+================+=========================+
| Children_Apps |      4.17       |       -0       |          15000          |
+---------------+-----------------+----------------+-------------------------+
| Teenager_Apps |      9.17       |       -0       |          25000          |
+---------------+-----------------+----------------+-------------------------+

Constraints Table:
+---------------------+---------------+----------------+------------------+---------+
|        Name         |  Final Value  |  Shadow Price  |  RHS Constraint  |  Slack  |
+=====================+===============+================+==================+=========+
|        Theme        |     78.33     |       -0       |       100        |  21.67  |
+---------------------+---------------+----------------+------------------+---------+
| Graphics_&_Imaging  |      240      |     277.78     |       240        |   -0    |
+---------------------+---------------+----------------+------------------+---------+
| Educational_Content |    303.33     |       -0       |       400        |  96.67  |
+---------------------+---------------+----------------+------------------+---------+
|       Coding        |      180      |      1250      |       180        |   -0    |
+---------------------+---------------+----------------+------------------+---------+
| Testing_&_Launching |     88.33     |       -0       |       120        |  31.67  |
+---------------------+---------------+----------------+------------------+---------+
** The constraints for which the Slack value is 0 are binding constraints
```

Note that in PuLP, we are unable to directly get the values for allowable increase/decrease for either the coefficients of the objective function or the RHS values of the constraints for the Sensitivity Report. These outputs are easier to obtain in Excel or by using one of the commercial Linear Programming in Python tools such as Gurobipy. The corresponding solution from Excel is shown in Chart 2.

Chart 2. Excel Solver Sensitivity Report



Microsoft Excel 16.0 Sensitivity Report

Variable Cells

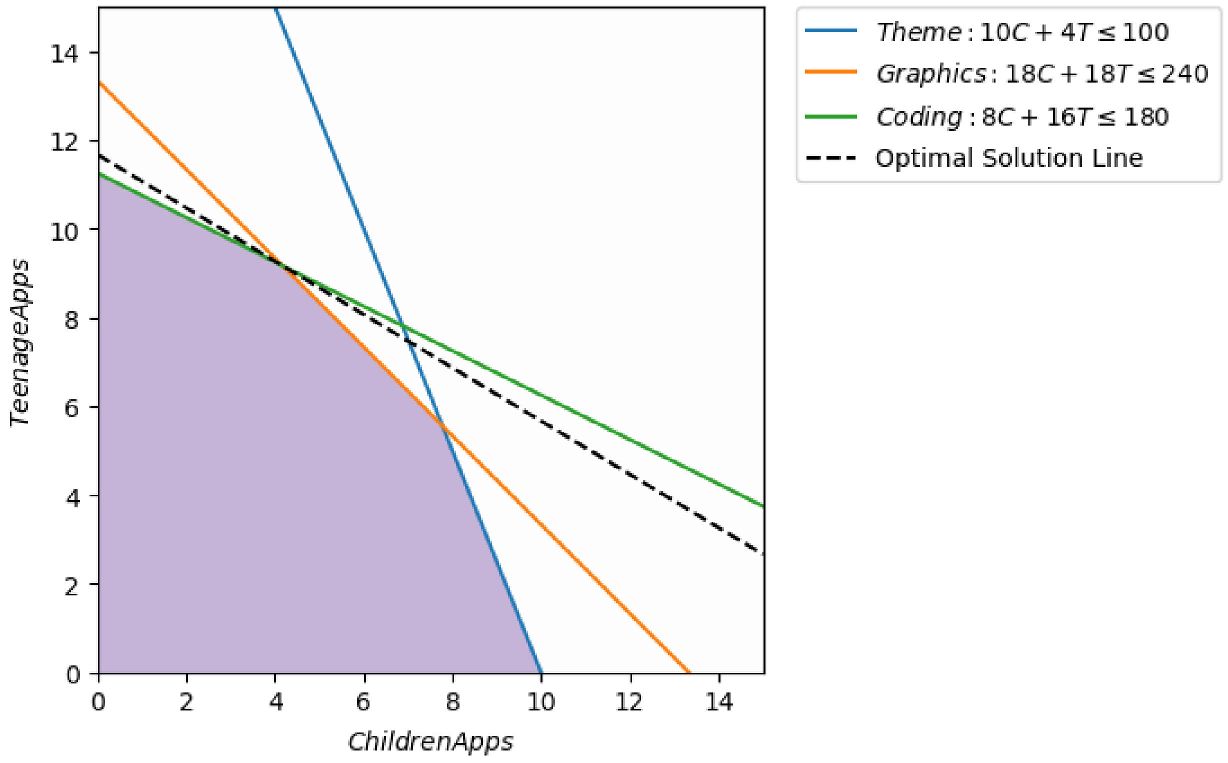
Cell	Name	Final Value	Reduced Cost	Objective Coefficient	Allowable Increase	Allowable Decrease
C12	Decision Variables C	4.17	0	15000	10000	2500
D12	Decision Variables T	9.17	0	25000	5000	10000

Constraints

Cell	Name	Final Value	Shadow Price	Constraint R.H. Side	Allowable Increase	Allowable Decrease
D17	Theme LHS	78.33	0.00	100	1E+30	21.67
D18	Graphics and Images LHS	240.00	277.78	240	24.38	37.50
D19	Educational Content LHS	303.33	0.00	400	1E+30	96.67
D20	Coding LHS	180.00	1250.00	180	33.33	28.89
D21	Testing and Launching LHS	88.33	0.00	120	1E+30	31.67

We can also visualize this maximum solution of $291,666.67 for 4.17 Children's Apps and 9.17 Teenager's Apps as seen in Chart 2. Note that the Graphics & Imaging and the Coding Constraints are binding (because the slack values are 0), so we are only keeping the constraints that form the region of feasibility in this chart.

Chart 3. Optimal Solution Line for PAR Insights, Inc. Profit Maximization Problem



This example is mainly to demonstrate how Python is able to create similar outputs as Excel for Linear Programming. There are more complexities that will be covered in future examples. Hope you've enjoyed learning alongside us… give our website www.analyticstx.com a visit and let us know if we can help you with any of your business or learning needs!

APPENDIX 1. PYTHON CODE TO CREATE THE GRAPHS FOR LP

Code for Creating Chart 1:

```
# Plot the feasible region
d = np.linspace(-2, 16, 300)
C, T = np.meshgrid(d,d)
plt.imshow( ((10*C + 4*T <= 100) & (18*C + 18*T <= 240) & (20*C + 24*T <= 400) &
(8*C + 16*T <= 180) & (8*C + 6*T <= 120)).astype(int),
                extent=(C.min(),C.max(),T.min(),T.max()),origin="lower",
cmap="Purples", alpha = 0.3);

# Plot the lines defining the constraints:
C=np.linspace(0, 16, 2000)
# 10*C + 4*T <= 100, "Theme"
T1 = (100-10*C)/4.0
# 18*C + 18*T <= 240, "Graphics & Imaging"
T2 = (240-18*C)/18.0
# 20*C + 24*T <= 400, "Educational Content"
T3 = (400-20*C)/24.0
# 8*C + 16*T <= 180, "Coding"
T4 = (180-8*C)/16.0
# 8*C + 6*T <= 120, "Testing & Launching"
T5 = (120-8*C)/6.0

#Make Plot
plt.plot(C,T1, label=r'$Theme: 10C+4T\leq100$')
plt.plot(C,T2, label=r'$Graphics: 18C+18T\leq240$')
plt.plot(C,T3, label=r'$Content: 20C+24T\leq400$')
plt.plot(C,T4, label=r'$Coding: 8C+16T\leq180$')
plt.plot(C,T5, label=r'$Testing: 8C+6T\leq120$')
plt.xlim(0, 15)
plt.ylim(0, 15)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.xlabel(r'$Children Apps$')
plt.ylabel(r'$Teenage Apps$')
```

Code for Creating Chart 3:

```
# plot the feasible region
d = np.linspace(-2, 16, 300)
C, T = np.meshgrid(d,d)
plt.imshow( ((10*C + 4*T <= 100) & (18*C + 18*T <= 240) & (20*C + 24*T <= 400) &
(8*C + 16*T <= 180) & (8*C + 6*T <= 120)).astype(int),
                extent=(C.min(),C.max(),T.min(),T.max()),origin="lower",
cmap="Purples", alpha = 0.3);

# Plot the lines defining the constraints:
C=np.linspace(0, 16, 2000)
# 10*C + 4*T <= 100, "Theme"
T1 = (100-10*C)/4.0
# 18*C + 18*T <= 240, "Graphics & Imaging"
T2 = (240-18*C)/18.0
# 8*C + 16*T <= 180, "Coding"
T4 = (180-8*C)/16.0
#Optimal Solution Maximize Z: 15000C + 25000T = 291666.67
Topt = (291666.67 - 15000*C)/25000.0

#Make Plot
plt.plot(C,T1, label=r'$Theme: 10C+4T\leq100$')
plt.plot(C,T2, label=r'$Graphics: 18C+18T\leq240$')
plt.plot(C,T4, label=r'$Coding: 8C+16T\leq180$')
plt.plot(C, Topt, label = "Optimal Solution Line", c='k', linestyle='dashed')
plt.xlim(0, 15)
plt.ylim(0, 15)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.xlabel(r'$Children Apps$')
plt.ylabel(r'$Teenage Apps$')
```

APPENDIX 2. PYTHON CODE FOR PRETTY_OUTPUT() FOR LP MODELS

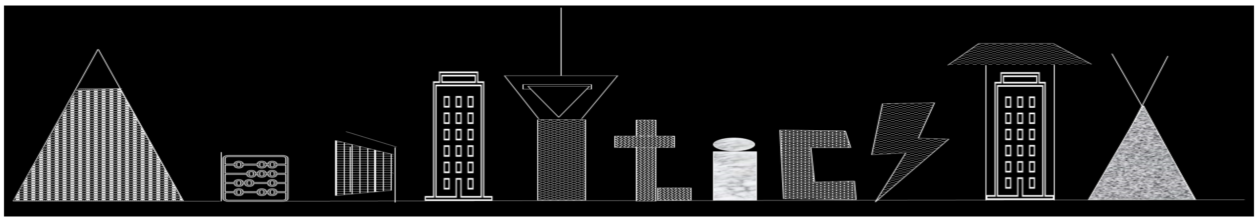
```
def pretty_output(model):
    # Variable Cells Table
    variable_cells_table = [["Name", "Optimal Value", "Reduced Cost", "Objective
Coefficient"]]
    variable_cells = [
        [variable, variable.varValue, variable.dj, model.objective.get(variable,
0.0)]
        for variable in model.variables()
    ]
    for row in variable_cells:
        variable_cells_table.append([row[0].name, f"{row[1]:.2f}",
f"{row[2]:.2f}", f"{row[3]:.2f}"])

    # Constraints Table
    constraints_table = [["Name", "Final Value", "Shadow Price", "RHS
Constraint", "Slack"]]
    constraints = [
        [constraint.name, (-constraint.constant -constraint.slack),
constraint.pi, -constraint.constant, constraint.slack]
        for constraint in model.constraints.values()
    ]
    for row in constraints:
        constraints_table.append([row[0], f"{row[1]:.2f}", f"{row[2]:.2f}",
f"{row[3]:.2f}", f"{row[4]:.2f}"])

    # Print the variable cells table and constraints table using tabulate
    print(model)
    print(f"Model status: {model.status}, {p.LpStatus[model.status]}")
    print("Value of Objective function:
{:,}".format(round(model.objective.value(), 2)))
    print("Variable Cells Table:")
    print(tabulate(variable_cells_table, headers='firstrow', tablefmt='grid',
numalign='center', stralign='center'))
    print("\nConstraints Table:")
    print(tabulate(constraints_table, headers='firstrow', tablefmt='grid',
numalign='center', stralign='center'))
    print("** The constraints for which the Slack value is 0 are binding
constraints")
```

ANALYTICS TX, LLC

Linear Programming Using Python – 3 Variable Example

Dr. Kruti Lehenbauer

4/18/2023



11703 Huebner Rd.| Ste 106-451| San Antonio, TX 78230 | Ph. 972-292-8338| ANALYTICSTX.COM

Example: Profit Maximization with 3 Variables

Continuing with the effort to learn (and share) Python Programming to perform various Data Analytics operations instead of using the traditional Excel and STATA methods that I have used for over 2 decades, here is the SECOND installment of solving a fairly simple linear programming profit maximization problem in Python. This example builds on the previous Example that I had shared and is also included in the book, *Introduction to Management Science: A Stepwise Approach to Basic Models in a Management Toolkit* (referred to as "the book" for the remainder of this file) and it is available in Kindle and Paperback format on Amazon from here.

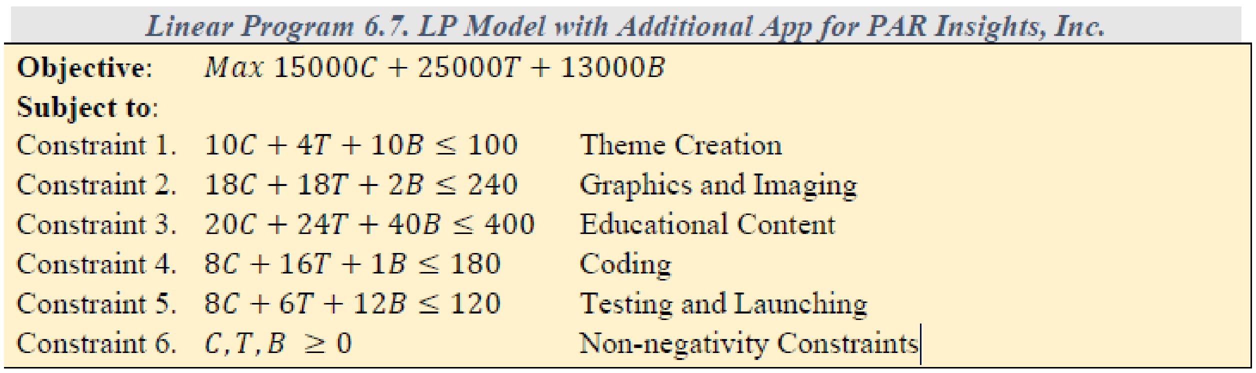
Chapter 6 of the book goes into detailed explanations of what Linear Programming is and how it can be used in business applications to optimize outcomes, so I am not delving too deeply into the background or the theory behind LP in this document. The book also covers step by step methods to solve LP Models using Graphical Methods and Excel Solver, so please feel free to refer to that if you need the stepwise approach.

In this example, we will be focusing on recreating and solving the Example 6.5 titled "PAR Insights, Inc Adding an App" from page 155-158 of the book.

If you followed along with attempting Example 6.1 from my previous notes, you do not need to import everything again, as long as you are using the same ipynb or py file. Otherwise, if you want start with a new Notebook, import Pandas and Tabulate as shown below. The most important import for solving Linear Programming problems in Python is PuLP[1]. I am also starting to use more compact command lines as compared to the previous example.

```
import pandas as pd, pulp as p
from tabulate import tabulate
```

The Linear Program that we need to solve is represented by the following LP Model (see pg. 156 of the book). Note that as compared to the previous problem, we are now trying to add another app for Babies in the production line for PAR Insights. The main reason for considering adding another app is that as we observed in the previous example, there was a lot of slack time available in some of the departments in the company and maximizing profits alongside maximizing usage of available resources is a key element of business.



Linear Program 6.7. LP Model with Additional App for PAR Insights, Inc.

Objective:	$Max\ 15000C + 25000T + 13000B$	
Subject to:		
Constraint 1.	$10C + 4T + 10B \leq 100$	Theme Creation
Constraint 2.	$18C + 18T + 2B \leq 240$	Graphics and Imaging
Constraint 3.	$20C + 24T + 40B \leq 400$	Educational Content
Constraint 4.	$8C + 16T + 1B \leq 180$	Coding
Constraint 5.	$8C + 6T + 12B \leq 120$	Testing and Launching
Constraint 6.	$C, T, B \geq 0$	Non-negativity Constraints

[1] You might need to install these packages before you can import them to the workspace. Note that I have skipped some output information in the code for the sake of brevity.

The new app requires 10 hours of Theme Department, 2 hours from Graphics & Imaging Department, 40 hours from the Educational Content Department, 1 hour from the Coding Department, and 12 hours from the Testing Department. Now, we can try to create this in Python in the following stepwise manner:

First, we put in some text to describe what the Problem is and set up the type of model that we want to solve in PuLp by "initializing" the model (PARM5 in our example) as shown below:

```
Example 6_5 PAR Insights - additional product to be created
# The Problem: PAR Insights is trying to expand its business into the educational
apps market and can create three types of products with which they can enter the
market.
# The management is trying to identify how many apps of each type should they
launch in the next four months after adding Baby Apps to the lists.
# Also, they want the answers in terms of Integers, and not continuous variables.

PARM5 = p.LpProblem("PAR_Insights_Profit_Max_3_products", p.LpMaximize)
```

Let us first create the datasets that will allow us to access the various hours, apps, profit values, and other relevant information that is available in the problem (see the book for details). We will create two datasets or "DataFrames" for ease of access.

```
# Data
idx = ['C','T','B']
df = pd.DataFrame({"Apps": idx,
     'Theme':[10, 4, 10],
     'Graphics':[18,18,2],
     'EdContent':[20, 24, 40],
     'Coding':[8, 16, 1],
     'Testing':[8, 6, 12],
     'Profit':[15000, 25000, 13000]}).set_index('Apps')
avail = pd.DataFrame({"Dept":['Theme','Graphics','EdContent',
'Coding','Testing'], "Hours":[100, 240, 400, 180, 120]}).set_index('Dept')
```

By using the commands "print(df)" and "print(avail)" the following output can be obtained:

```
      Theme  Graphics  EdContent  Coding  Testing  Profit
Apps
C        10        18         20       8        8   15000
T         4        18         24      16        6   25000
B        10         2         40       1       12   13000
           Hours
Dept
Theme        100
Graphics     240
EdContent    400
Coding       180
Testing      120
```

In the Linear Program described, there are three main variables that need to be estimated: the number of Children's Apps (C), the number of Teenager's Apps (T), and the number of Baby Apps (B). In our data named "df", these three variables are the "Apps" and are assigned as "indexes" for the dataframe. Since they are in the DataFrame, we can define the Decision Variables in Python, with one line. Notice we are also putting the requirement that these variables can only take integer values by choosing "cat" (represents category) to be 'Integer.'

```
# Decision Variables
x = p.LpVariable.dicts('',df.index, lowBound=0, cat='Integer')
```

The p.LpVariable assigns the variable names and their descriptions to the LP Model. Next, we will define the Objective Function (the function that we need to maximize) and the Constraints that need to be added to our model (PARM5) as shown below. Note that unlike the previous example, we are using more complex methods to write the code but it implies that the code is shorter and more consistent.

```
#Objective Function
PARM5 += sum([x[idx]*df['Profit'][idx] for idx in df.index])
# Constraints
for dept in avail.index:
    PARM5 += sum([x[idx]*df[dept][idx] for idx in df.index]) <=
avail['Hours'][dept], f"{dept} Department"
```

Since there are three variables in this example, we can no longer visualize the Linear Problem in a 2-dimensional graph. The concept remains the same. The constraints create a region of feasibility and the optimal solution occurs at one of the end-points of this region in a multi-dimensional space. Now, we can solve the problem by using the "solve" function. There are many ways to see the relevant output. I have defined a special function for output called "pretty_output()" and the code can be found in Appendix 1 below.

```
# Solving the model:
PARM5sol = PARM5.solve()
pretty_output(PARM5)
```

This will yield the following output below the cell (if you are using ipynb Jupyter notebook).

```
PAR_Insights_Profit_Max_3_products:
MAXIMIZE
13000*_B + 15000*_C + 25000*_T + 0
SUBJECT TO
Theme_Department: 10 _B + 10 _C + 4 _T <= 100
Graphics_Department: 2 _B + 18 _C + 18 _T <= 240
EdContent_Department: 40 _B + 20 _C + 24 _T <= 400
Coding_Department: _B + 8 _C + 16 _T <= 180
Testing_Department: 12 _B + 8 _C + 6 _T <= 120

VARIABLES
0 <= _B Integer
0 <= _C Integer
0 <= _T Integer

Model status: 1, Optimal
Value of Objective function: 319,000.0
Variable Cells Table:
```

```
+--------+-----------------+----------------+-------------------------+
|  Name  |  Optimal Value  |  Reduced Cost  |  Objective Coefficient  |
+========+=================+================+=========================+
|   _B   |        3        |     13000      |          13000          |
+--------+-----------------+----------------+-------------------------+
|   _C   |        2        |     15000      |          15000          |
+--------+-----------------+----------------+-------------------------+
|   _T   |       10        |     25000      |          25000          |
+--------+-----------------+----------------+-------------------------+

Constraints Table:
+----------------------+---------------+----------------+------------------+---------+
|         Name         |  Final Value  |  Shadow Price  |  RHS Constraint  |  Slack  |
+======================+===============+================+==================+=========+
|   Theme_Department   |      90       |       -0       |       100        |   10    |
+----------------------+---------------+----------------+------------------+---------+
| Graphics_Department  |      222      |       -0       |       240        |   18    |
+----------------------+---------------+----------------+------------------+---------+
| EdContent_Department |      400      |       -0       |       400        |   -0    |
+----------------------+---------------+----------------+------------------+---------+
|  Coding_Department   |      179      |       -0       |       180        |    1    |
+----------------------+---------------+----------------+------------------+---------+
|  Testing_Department  |      112      |       -0       |       120        |    8    |
+----------------------+---------------+----------------+------------------+---------+
** The constraints for which the Slack value is 0 are binding constraints
```

As one can see the optimal number of Baby Apps to make are 3 for a profit of $13,000 each, 2 Children's Apps for profit of $15,000 each and 10 Teenager Apps for profit of $25,000 each for a a total maximum profit of $319,000 in the four months that are available to PAR Insights, Inc.

Hope you are enjoyng learning Linear Programming in Python alongside us… give our website www.analyticstx.com a visit and let us know if we can help you with any of your business or learning needs!

APPENDIX 1. PYTHON CODE FOR PRETTY_OUTPUT() FOR LP MODELS

```
def pretty_output(model):
    # Variable Cells Table
    variable_cells_table = [["Name", "Optimal Value", "Reduced Cost", "Objective
Coefficient"]]
    variable_cells = [
        [variable, variable.varValue, variable.dj, model.objective.get(variable,
0.0)]
        for variable in model.variables()
    ]
    for row in variable_cells:
        variable_cells_table.append([row[0].name, f"{row[1]:.2f}",
f"{row[2]:.2f}", f"{row[3]:.2f}"])

    # Constraints Table
    constraints_table = [["Name", "Final Value", "Shadow Price", "RHS
Constraint", "Slack"]]
    constraints = [
        [constraint.name, (-constraint.constant -constraint.slack),
constraint.pi, -constraint.constant, constraint.slack]
        for constraint in model.constraints.values()
    ]
    for row in constraints:
        constraints_table.append([row[0], f"{row[1]:.2f}", f"{row[2]:.2f}",
f"{row[3]:.2f}", f"{row[4]:.2f}"])

    # Print the variable cells table and constraints table using tabulate
    print(model)
    print(f"Model status: {model.status}, {p.LpStatus[model.status]}")
    print("Value of Objective function:
{:,}".format(round(model.objective.value(), 2)))
    print("Variable Cells Table:")
    print(tabulate(variable_cells_table, headers='firstrow', tablefmt='grid',
numalign='center', stralign='center'))
    print("\nConstraints Table:")
    print(tabulate(constraints_table, headers='firstrow', tablefmt='grid',
numalign='center', stralign='center'))
    print("** The constraints for which the Slack value is 0 are binding
constraints")
```

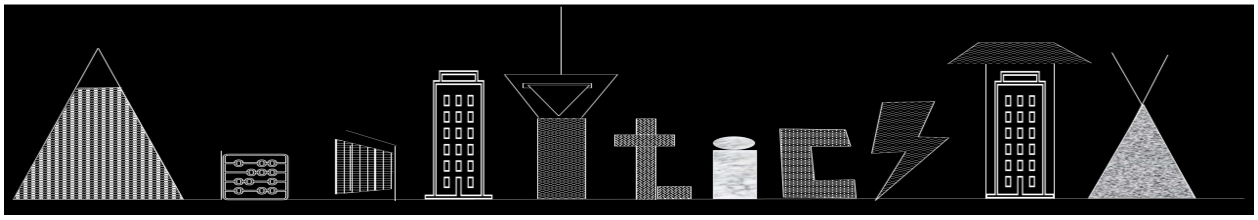
Linear Programming Example 3

ANALYTICS TX, LLC

Linear Programming Using Python – Shortest Route Example

Dr. Kruti Lehenbauer

4/28/2023



11703 Huebner Rd.| Ste 106-451| San Antonio, TX 78230 | Ph. 972-292-8338| ANALYTICSTX.COM

Example: Shortest Route Transportation Problem

Continuing with the effort to learn (and share) Python Programming to perform various Data Analytics operations instead of using the traditional Excel and STATA methods that I have used for over 2 decades, here is the THIRD installment of solving a linear programming SHORTEST ROUTE problem in Python. This example is included in Chapter 8 of the book, *Introduction to Management Science: A Stepwise Approach to Basic Models in a Management Toolkit* (referred to as "the book" for the remainder of this file) and it is available in Kindle and Paperback format on Amazon from here.

Chapter 8 of the book goes into detailed explanations of usage of Linear Programming (LP) in Supply Chain and Distribution Models, so I am not delving too deeply into the background or the theory behind the LP in this document. The book also covers step by step methods to solve LP Models using the Excel Solver, so please feel free to refer to that if you need the stepwise approach to solve the problem in Excel.

In this document, we will be focusing on recreating and solving the Example 8.4 titled "Shortest Route Problem BCWM" on pages 236-240 of the book.

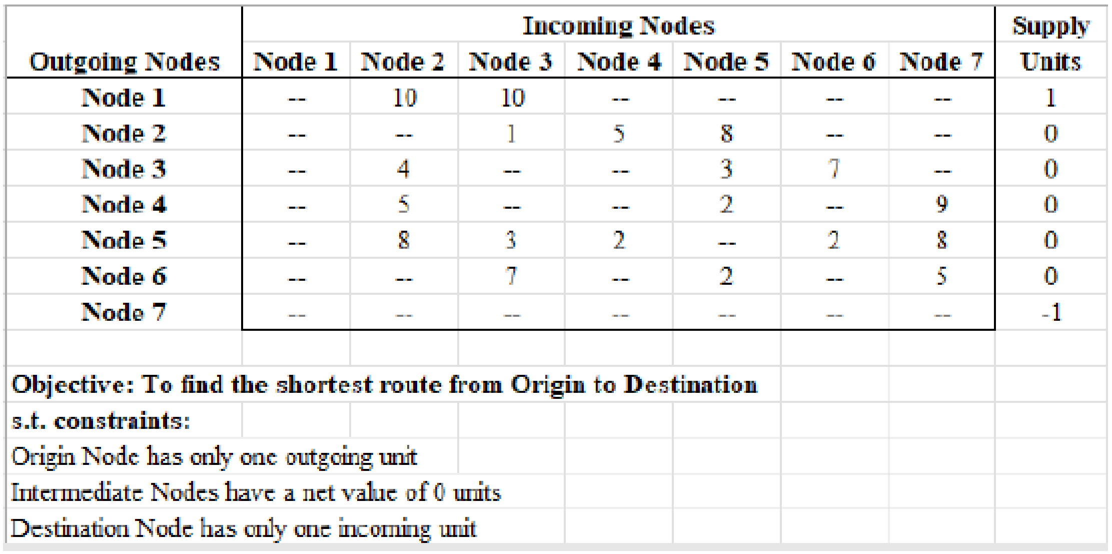
On a new Jupyter Notebook (extension is .ipynb) or Python program, import the following[1]:

- Pandas – for data operations
- PuLP – for Linear Programming
- Networkx – for creating Network graphs for the route or flow
- MatPlotLib.PyPlot – to work with Networkx to create better graphical output
- Tabulate – for creating table for LP output (optional if you want to skip the output)

```
import pandas as pd, pulp as p, networkx as nx, matplotlib.pyplot as plt
from tabulate import tabulate
```

The LP for finding the shortest route that we need to solve is represented by the following LP Model (see pg. 237-239 of the book for details on how to create the constraints and objective functions). Here, the goal is to minimize the total number of miles travelled by a trash truck in one trip going from the Start Node to the final destination or End Node via intermediate nodes that represent various neighborhoods. The data that is being used to create this model is shown alongside the LP Model for ease of reference in this document. The book discusses the intermediate steps in significant details so those are not being repeated in this document.

[1] You might need to install these packages before you can import them to the workspace using the pip install method in your command window. Note that I have skipped some output information in the code for the sake of brevity.



	Incoming Nodes							Supply
Outgoing Nodes	**Node 1**	**Node 2**	**Node 3**	**Node 4**	**Node 5**	**Node 6**	**Node 7**	**Units**
Node 1	--	10	10	--	--	--	--	1
Node 2	--	--	1	5	8	--	--	0
Node 3	--	4	--	--	3	7	--	0
Node 4	--	5	--	--	2	--	9	0
Node 5	--	8	3	2	--	2	8	0
Node 6	--	--	7	--	2	--	5	0
Node 7	--	--	--	--	--	--	--	-1

Objective: To find the shortest route from Origin to Destination
s.t. constraints:
Origin Node has only one outgoing unit
Intermediate Nodes have a net value of 0 units
Destination Node has only one incoming unit

Linear Program 8.8. Shortest Route LP for BCWM

Objective:

$$Min\ Miles = 10N_{12} + 10N_{13} + 1N_{23} + 5N_{24} + 8N_{25} + 1N_{32} + 3N_{35} + 7N_{36} + 5N_{42} + 2N_{45} + 9N_{47} + 8N_{52} + 3N_{53} + 2N_{54} + 2N_{56} + 8N_{57} + 7N_{63} + 2N_{65} + 5N_{67}$$

Subject to:

Constraint 1.	$N_{12} + N_{13} = 1$	Origin Node 1
Constraint 2.	$N_{23} + N_{24} + N_{25} - (N_{12} + N_{32} + N_{42} + N_{52}) = 0$	Node 2
Constraint 3.	$N_{32} + N_{35} + N_{36} - (N_{13} + N_{23} + N_{53} + X_{63}) = 0$	Node 3
Constraint 4.	$N_{42} + N_{45} + N_{47} - (N_{24} + N_{54}) = 0$	Node 4
Constraint 5.	$N_{52} + N_{53} + N_{54} + N_{56} + N_{57} - (N_{25} + N_{35} + N_{45} + N_{65}) = 0$	Node 5
Constraint 6.	$N_{63} + N_{65} + N_{67} - (N_{36} + N_{56}) = 0$	Node 6
Constraint 7.	$N_{47} + N_{57} + N_{67} = -1$	Destination Node 7
Constraint 8.	$N_{ij} \geq 0$	Non-negativity constraints

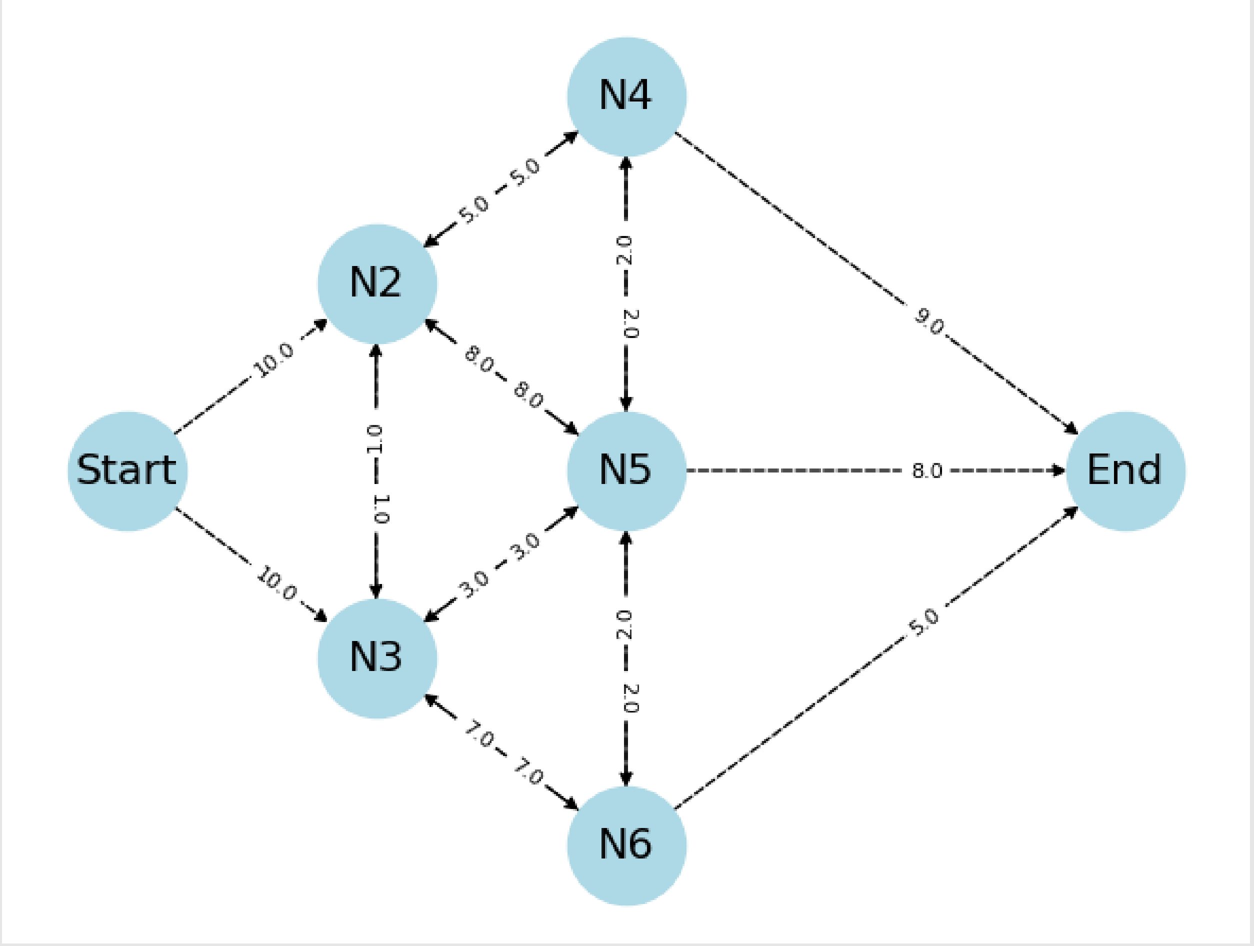
What is most important about this problem is that only one Trash Truck is entering this network at a given time and thus, the value of each of the variables or pathways is either 1 or 0. The 1 represents that the truck takes a particular route, whereas the 0 indicates that it is not efficient for the truck to take that route. Thus, we have binary variables in our LP, which is different than the continuous or integer variables that we used in earlier examples. Now, we can try to create the example in Python in the following stepwise manner:

First, since this is a Network Problem, the easiest way to visualize the problem at hand is to create a Network graph after importing the data. Note that the data can be imported from an excel file or a csv file as shown below. We are also setting the Index of our dataset to the "Nodes" function for ease of use in programming. This creates our DataFrame for the Graph as well as the LP. The 'NaN' values in the DataFrame refer to "Not a Number" which is standard practice in Python for missing values. Now, a Network graph function can be created to get the output in the form of a graph. Appendix 1 below shows how to define this function in Python. It takes the arguments "df" that represents the name of the dataset and "num", which represents whether it is the initial graph (1) or post-LP graph (2).

```
# Example 8.4: Shortest Route Problem - Binary Variables - 'SR'
# The Problem: Bexar County Waste Management Company manager needs to identify
the shortest route from the holding center to the dump in order to submit a
successful bid for purchasing a large-capacity but fuel-inefficient trash truck.
# Get Data
df = pd.read_csv('Example8_4.csv').set_index('Nodes')
print(df)
# Draw Initial Network using "SR" and choose number=1 to make sure all possible
routes are included
Network_graph_SR(df,1)
```

OUTPUT:

```
       Start    N2    N3   N4   N5   N6  End  Supply
Nodes
Start    NaN  10.0  10.0  NaN  NaN  NaN  NaN       1
N2       NaN   NaN   1.0  5.0  8.0  NaN  NaN       0
N3       NaN   1.0   NaN  NaN  3.0  7.0  NaN       0
N4       NaN   5.0   NaN  NaN  2.0  NaN  9.0       0
N5       NaN   8.0   3.0  2.0  NaN  2.0  8.0       0
N6       NaN   NaN   7.0  NaN  2.0  NaN  5.0       0
End      NaN   NaN   NaN  NaN  NaN  NaN  NaN      -1
```



The next step is to initialize the Linear Program (Trash in our example) by using PuLP, as shown below:

```
# Define LP using a function
Trash = p.LpProblem("Example8_4", p.LpMinimize)
```

In this example, the variables go in multiple directions. That is, there are valid routes from one node to another and then there are some invalid routes (note the NaN values in the DataFrame). Only the valid routes should be considered for defining the variables, defining the Objective Function, and defining the Constraints. We also know that a Truck can only LEAVE from the "Start" Node and stops at the "End" Node. We also need to include the requirement that these variables can only take binary (0, 1) values by choosing "cat" (represents category) to be 'Binary' when defining the variables.

This implies that defining a function that iterates over various Nodes and connections in the DataFrame might be the most efficient method to define each of the variables and constraints. The following code helps us to define this specific function (please note that the text is wrapping automatically in the following code and you might have to adjust it a bit in your Python Notebook to ensure that the code is in compliance with the program requirements):

```
def add_variables_and_constraints(df, model):
    x = {}
    for i in df.index:
        for j in df.columns[:-1]:
            if not pd.isna(df.loc[i,j]):
                x[(i,j)] = p.LpVariable(f'{i}{j}', lowBound=0, cat="Binary")
    # Add constraints
    for i in df.index:
        if i == "Start":
            model += sum(x[i,j] for j in df.columns[:-1] if not
pd.isna(df.loc[i,j])) == 1, f'Start Constraint'
        elif i in df.index[1:6]:
            model += sum(x[i,j] for j in df.columns[:-1] if not
pd.isna(df.loc[i,j])) == sum(x[j,i] for j in df.columns[:-1] if not
pd.isna(df.loc[j,i])), f'Node_{i} Constraint'
    model += sum(x[i,'End'] for i in df.index if not pd.isna(df.loc[i,j])) == 1,
f'End Constraint'
    # Add objective function
    model += sum(x[i,j]*df[j][i] for i in df.index for j in df.columns[:-1] if
not pd.isna(df.loc[i,j]))
    return x
# Adding Variables and Constraints to LP Problem
x = add_variables_and_constraints(df, Trash)
print(Trash)
```

If we print out the LP Program by using the command "print(Trash)" hereafter, we would get the following output. This allows us to cross-check the code variables and equations with our originally derived equations. Note that there is a small difference in the "End Constraint" in the use of the "minus" sign on the RHS. This is because of the way PuLP arranges its variables as compared to how they are arranged in Excel Solver.

```
Example8_4:
MINIMIZE
1.0*N2N3 + 5.0*N2N4 + 8.0*N2N5 + 1.0*N3N2 + 3.0*N3N5 + 7.0*N3N6 + 9.0*N4End +
5.0*N4N2 + 2.0*N4N5 + 8.0*N5End + 8.0*N5N2 + 3.0*N5N3 + 2.0*N5N4 + 2.0*N5N6 +
5.0*N6End + 7.0*N6N3 + 2.0*N6N5 + 10.0*StartN2 + 10.0*StartN3 + 0.0
SUBJECT TO
Start_Constraint: StartN2 + StartN3 = 1
Node_N2_Constraint: N2N3 + N2N4 + N2N5 - N3N2 - N4N2 - N5N2 - StartN2 = 0
Node_N3_Constraint: - N2N3 + N3N2 + N3N5 + N3N6 - N5N3 - N6N3 - StartN3 = 0
Node_N4_Constraint: - N2N4 + N4End + N4N2 + N4N5 - N5N4 = 0
Node_N5_Constraint: - N2N5 - N3N5 - N4N5 + N5End + N5N2 + N5N3 + N5N4 + N5N6 -
N6N5 = 0
Node_N6_Constraint: - N3N6 - N5N6 + N6End + N6N3 + N6N5 = 0
End_Constraint: N4End + N5End + N6End = 1

VARIABLES
0 <= N2N3 <= 1 Integer
0 <= N2N4 <= 1 Integer
0 <= N2N5 <= 1 Integer
0 <= N3N2 <= 1 Integer
0 <= N3N5 <= 1 Integer
0 <= N3N6 <= 1 Integer
0 <= N4End <= 1 Integer
0 <= N4N2 <= 1 Integer
0 <= N4N5 <= 1 Integer
0 <= N5End <= 1 Integer
0 <= N5N2 <= 1 Integer
0 <= N5N3 <= 1 Integer
0 <= N5N4 <= 1 Integer
0 <= N5N6 <= 1 Integer
0 <= N6End <= 1 Integer
0 <= N6N3 <= 1 Integer
0 <= N6N5 <= 1 Integer
0 <= StartN2 <= 1 Integer
0 <= StartN3 <= 1 Integer
```

Note that since the variables are all defined as being "Binary", the LP places their values as being between 0 and 1 and imposes the "Integer" restriction on them implying that each variable can ONLY take one of the two values of 0 and 1 in the solution. The full program has been defined using the function defined above, so now we can proceed to solve it by using the command "Trash.solve()" which calls the default solver from the PuLP module. In order to get output for all variables, we will use a modified version of the pretty_output() function that we used in previous examples called the "transport_output()" which is included below.

```
Trash_sol = Trash.solve()
def transport_output(df,model):
    # Variable Cells Table
    variable_cells_table = [["Name", "Optimal Value"]]
    variable_cells = [[variable, variable.varValue] for variable in
model.variables()]
    for row in variable_cells:
        variable_cells_table.append([row[0].name, format(row[1], ',.2f')])
    # Print the variable cells table using tabulate to open file (switch # in
line 12)
    print(df, model,
        f"Model status: {model.status}, {p.LpStatus[model.status]}",
        "Value of Objective function: {:,}".format(round(model.objective.value(),
2)),
        "Variable Cells Table:",
        tabulate(variable_cells_table, headers='firstrow', tablefmt='grid',
numalign='center', stralign='center'),sep ="\n")
transport_output(df,Trash)
```

The produced output is shown below. Note that I am not including the "df" or the "model" output since they were shared earlier.

```
Model status: 1, Optimal
Value of Objective function: 20.0
Variable Cells Table:
+---------+-----------------+
|  Name   |  Optimal Value  |
+=========+=================+
|  N2N3   |        0        |
+---------+-----------------+
|  N2N4   |        0        |
+---------+-----------------+
|  N2N5   |        0        |
+---------+-----------------+
|  N3N2   |        0        |
+---------+-----------------+
|  N3N5   |        1        |
+---------+-----------------+
|  N3N6   |        0        |
+---------+-----------------+
|  N4End  |        0        |
+---------+-----------------+
|  N4N2   |        0        |
+---------+-----------------+
|  N4N5   |        0        |
+---------+-----------------+
|  N5End  |        0        |
+---------+-----------------+
|  N5N2   |        0        |
+---------+-----------------+
|  N5N3   |        0        |
+---------+-----------------+
|  N5N4   |        0        |
+---------+-----------------+
|  N5N6   |        1        |
+---------+-----------------+
|  N6End  |        1        |
+---------+-----------------+
|  N6N3   |        0        |
+---------+-----------------+
|  N6N5   |        0        |
+---------+-----------------+
| StartN2 |        0        |
+---------+-----------------+
| StartN3 |        1        |
+---------+-----------------+
```

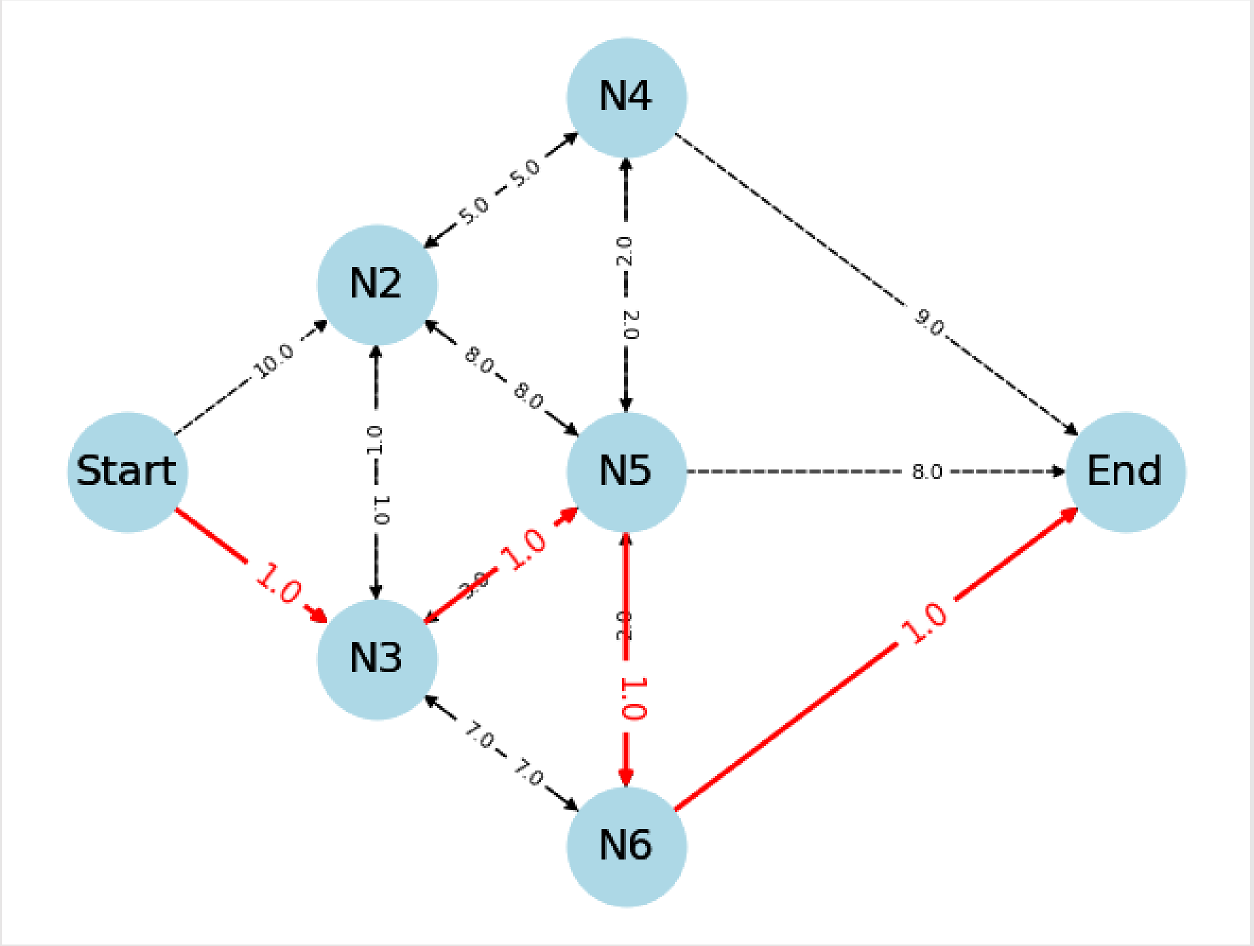
While this output is consistent with the output we obtained using Excel Solver (see pg. 239 of the book), it is not easy to interpret visually. The objective function is minimized at the value of 20, implying that the Trash Truck has to travel a minimum distance of 20 miles through the route "Start – Node3 – Node5 – Node6 – End.

In order to make this output more "readable" and to create an updated network, we can write some more code to clean up the results and put them in a new DataFrame called "res" which we can use to create the updated Network Graph. If you creating all the code in one box in the Jupyter Notebook, you do not need to include the first graph (as shown below) again..

```
# Creating Formatted Output from Linear Program
res = pd.DataFrame(index=df.index,columns=df.columns[:-1])
res_dict = {}
for i, j in x.keys():
    res_dict[i,j] = (x[i,j].varValue)
for i, row_label in enumerate(res.index):
    for j, col_label in enumerate(res.columns):
        res.loc[row_label, col_label] = res_dict.get((row_label, col_label), 
None)
res['Outgoing'] = res.sum(axis=1)
Network_graph_SR(df,1)
Network_graph_SR(res,2)
res.loc['Incoming'] = res.sum(axis=0)
res.loc[res.index[-1], res.columns[-1]] = ''
print(res)
```

The output from this code will be obtained directly under the box of the code in the Jupyter Notebooks. All of these outputs can also be written to an external file and graphs can be saved as .png or other formats if desired by adding appropriate lines of code. I have not included those lines here, but feel free to reach out if you need help with saving graphs or output files. The red pathway shows the exact route that the Trash Truck will take from the starting point to the ending point while collecting trash from the neighborhoods (nodes) that it drives through.

```
          Start    N2    N3    N4    N5    N6   End Outgoing
Nodes
Start      None   0.0   1.0  None  None  None  None      1.0
N2         None  None   0.0   0.0   0.0  None  None      0.0
N3         None   0.0  None  None   1.0   0.0  None      1.0
N4         None   0.0  None  None   0.0  None   0.0      0.0
N5         None   0.0   0.0   0.0  None   1.0   0.0      1.0
N6         None  None   0.0  None   0.0  None   1.0      1.0
End        None  None  None  None  None  None  None        0
Incoming      0   0.0   1.0   0.0   1.0   1.0   1.0
```



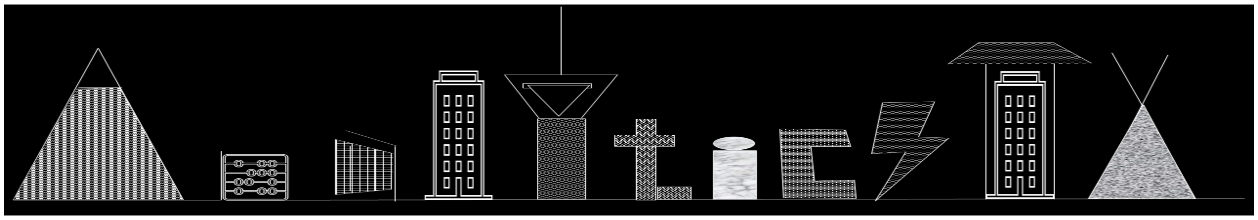
Hope you are enjoyng learning Linear Programming in Python alongside us… give our website www.analyticstx.com a visit and let us know if we can help you with any of your business or learning needs!

APPENDIX 1. PYTHON CODE FOR Network_graph_SR(df, num)

```
# Always double check all nodes for examples in the first few lines
# "SR": Shortest Route
def Network_graph_SR(df,num):
    start_nodes = list(df.index[0:1])
    inter_nodes = list(df.columns[1:-2])
    end_nodes = list(df.index[-1:])
    node = list(df.index)
    # print(start_nodes, inter_nodes, end_nodes)
    #Draw Graph
    G = nx.DiGraph()
    #Add nodes
    G.add_nodes_from(node)
    # print(G.nodes)
    #Add edges & Labels
    edge_labels={}
    for start_node in node:
        for end_node in node:
            if end_node in df.index:
                weight = df.loc[start_node, end_node]
                if weight != 0.0:
                    if not pd.isna(weight):
                        G.add_edge(start_node, end_node)
                        edge_labels[(start_node, end_node)] = weight
    # print(edge_labels)
    pos = {}
    pos.update((node, (-0.5, 0.5-index)) for index, node in
enumerate(inter_nodes[0:2]))
    pos.update((node, (0, 1-index)) for index, node in
enumerate(inter_nodes[2:]))
    pos.update((node, (-1,index)) for index, node in enumerate(start_nodes))
    pos.update((node, (1, index)) for index, node in enumerate(end_nodes))
    if num==1:
        nx.draw(G, pos, with_labels=True, node_size=2000, font_size=16,
node_color="lightblue", style ="dashed")
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
label_pos=0.4, font_size=8)
    else:
        nx.draw(G, pos, with_labels=True, node_size=2000, font_size=16,
node_color="lightblue", edge_color='red', width=2.0)
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
label_pos=0.4, font_size=12,font_color="red")
    plt.savefig("Graph_Example8_4" + str(num)+ "SR.png")
```

We Help You Make Sense of Your Data

ANALYTICS TX, LLC



Helping Businesses with Their Data Analytics Needs Since 2012