**CLARKE AND WRIGHT**

import numpy as np

import operator

from IPython.display import display

import pandas as pd

#Clarke-Wright Savings Algorithm

class CWSA(object):

'''

argument:

create an object with 'distances' attributes.

self.distances[(from\_node,to\_node)] = distance

output:

'''

def \_\_init\_\_(self):

self.distances = {}

def add\_dist(self, from\_node,to\_node,distance):

if from\_node != 'DC' and to\_node != 'DC':

if from\_node < to\_node:

self.distances[(from\_node,to\_node)] = distance

else:

self.distances[(to\_node,from\_node)] = distance

elif from\_node == 'DC':

self.distances[(to\_node,from\_node)] = distance

elif to\_node == 'DC':

self.distances[(from\_node,to\_node)] = distance

def CWSA\_dist\_matrix(cwsa):

'''

argument: cwsa object

output:

CWSA\_mtx (numpy array): rows = from\_node

columns = to\_node

entries = distance (above diagonal element)

diagonal and below diagonal elements are 0

'''

from\_list = []

dist\_dict = cwsa.distances

for from\_node,to\_node in dist\_dict:

if from\_node not in from\_list:

from\_list.append(from\_node)

from\_list.sort()

CWSA\_mtx = np.zeros((len(from\_list),len(from\_list)+1))

for from\_node,to\_node in dist\_dict:

if to\_node != 'DC':

CWSA\_mtx[from\_node-1,to\_node-1] = dist\_dict[(from\_node,to\_node)]

else:

CWSA\_mtx[from\_node-1,-1] = dist\_dict[(from\_node,to\_node)]

return CWSA\_mtx

def CWSA\_savings(cwsa):

'''

Given cwsa object, provide savings and distance table of

argument:

(object): cwsa object with complete distances attribute added by add\_dist

function

output:

CWSA\_dict(dataframe): 1st column = index

2nd column = (from\_node,to\_node)

3rd column = distance/cost saving for these nodes

CWSA\_mtx (dataframe): distance/cost (above diagonal element) and

saving (below diagonal element) of each pair of nodes

'''

CWSA\_mtx = CWSA\_dist\_matrix(cwsa)

CWSA\_dict = {}

for i in range(np.shape(CWSA\_mtx)[0]):

for j in range(i+1,np.shape(CWSA\_mtx)[0]):

saving = CWSA\_mtx[i,-1] + CWSA\_mtx[j,-1] - CWSA\_mtx[i,j]

CWSA\_mtx[j,i] = saving

CWSA\_dict[(i+1,j+1)] = saving

CWSA\_list = sorted(CWSA\_dict.items(),key=operator.itemgetter(1),

reverse=True)

CWSA\_savings\_df = pd.DataFrame(CWSA\_list)

CWSA\_df = pd.DataFrame(CWSA\_mtx)

return CWSA\_df,CWSA\_savings\_df

from IPython.display import display

cwsa = CWSA()

cwsa.add\_dist(1,2,16.3)

cwsa.add\_dist(1,3,16.5)

cwsa.add\_dist(1,4,20)

cwsa.add\_dist(1,5,19.6)

cwsa.add\_dist(1,6,17.9)

cwsa.add\_dist(1,7,9.3)

cwsa.add\_dist(1,'DC',12.7)

cwsa.add\_dist(2,3,7.2)

cwsa.add\_dist(2,4,14.9)

cwsa.add\_dist(2,5,16.6)

cwsa.add\_dist(2,6,16.6)

cwsa.add\_dist(2,7,12.7)

cwsa.add\_dist(2,'DC',11.5)

cwsa.add\_dist(3,4,8.9)

cwsa.add\_dist(3,5,10.1)

cwsa.add\_dist(3,6,11)

cwsa.add\_dist(3,7,10.8)

cwsa.add\_dist(3,'DC',9.8)

cwsa.add\_dist(4,5,7.3)

cwsa.add\_dist(4,6,13.4)

cwsa.add\_dist(4,7,19.1)

cwsa.add\_dist(4,'DC',17.5)

cwsa.add\_dist(5,6,12.9)

cwsa.add\_dist(5,7,16.4)

cwsa.add\_dist(5,'DC',16.1)

cwsa.add\_dist(6,7,9.4)

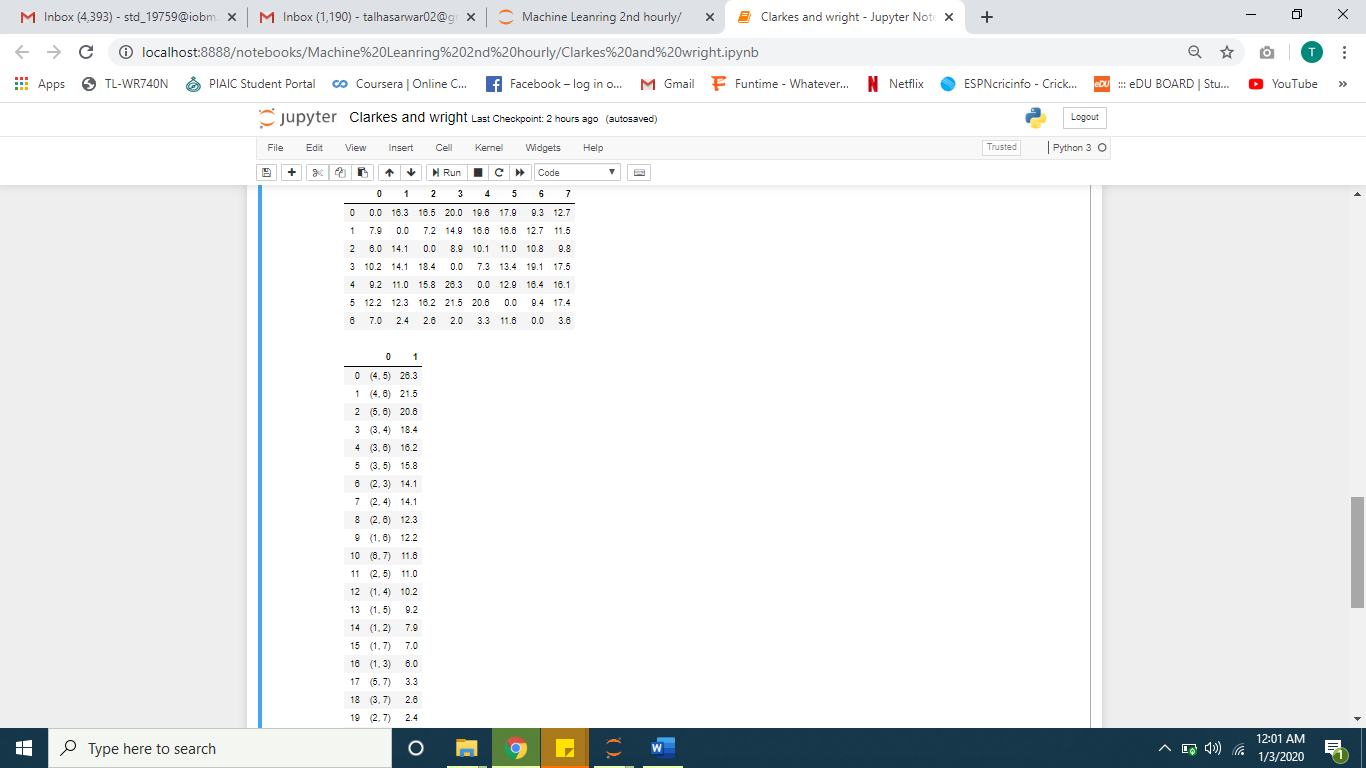
cwsa.add\_dist(6,'DC',17.4)

cwsa.add\_dist(7,'DC',3.6)

CWSA\_df, CWSA\_savings\_df = CWSA\_savings(cwsa)

display(CWSA\_df)

display(CWSA\_savings\_df)



HOLMES AND PARKER print ('Holmes and parker - route with the greatest saving')  
print (' Stop\n',

CWSA\_savings\_df.loc[0,0], 'Saving: ', CWSA\_savings\_df.loc[0,1], '\n',

CWSA\_savings\_df.loc[2,0], 'Saving: ', CWSA\_savings\_df.loc[0,1]+CWSA\_savings\_df.loc[2,1],'\n',

'Stop\n',

CWSA\_savings\_df.loc[6,0], 'Saving: ', CWSA\_savings\_df.loc[0,1]+CWSA\_savings\_df.loc[2,1]+CWSA\_savings\_df.loc[6,1],'\n',

CWSA\_savings\_df.loc[16,0], 'Saving: ', CWSA\_savings\_df.loc[0,1]+CWSA\_savings\_df.loc[2,1]+CWSA\_savings\_df.loc[6,1]+CWSA\_savings\_df.loc[16,1],'\n',

CWSA\_savings\_df.loc[15,0], 'Saving: ', CWSA\_savings\_df.loc[0,1]+CWSA\_savings\_df.loc[2,1]+CWSA\_savings\_df.loc[6,1]+CWSA\_savings\_df.loc[16,1]+CWSA\_savings\_df.loc[15,1],'\n',

'Stop\n'

)

