Evaluating the Solutions for the Logistic Problem in Alibaba

Note:Please make sure that you are using unique codes because this problem is different with previous midterm projects.

First lets generate the 300 random knapsack sizes and write the results to a csv

In [1]:
```
'''
Author: James Liao
Date: 3/ April/2021
ID: 490851
'''


#----------------
# Import packages
#----------------
import numpy as np
import scipy
import scipy.stats as stats
import csv



'''you will generate 300 capacities for each instance. It will be between 100-160

#-----------------------------------------------------------------
# Generate capacities with following restriction and export to csv
#-----------------------------------------------------------------
np.random.seed(1)
mean = 130
std = 30
lower = 100
upper = 160
N = 300
samples = scipy.stats.truncnorm.rvs((lower-mean)/std,(upper-mean)/std,loc=mean,sc
cap = samples.round().astype(int) # cap represents capacity
print(cap)


f = open("Knap_Sizes.csv", "w") # We will create this file.
for i in cap:
    f.write("%s\n" %(i))


f.close()
```

```
[126 142 100 120 111 107 113 122 125 132 126 140 114 151 102 139 126 133
 110 114 146 157 120 140 151 152 106 103 112 151 107 126 157 132 140 120
 140 148 102 143 159 143 118 146 108 127 153 119 119 110 102 139 115 118
 130 104 134 111 135 140 108 126 140 126 104 132 139 131 156 134 153 110
 110 147 125 112 154 122 143 142 151 136 143 122 118 152 126 157 139 136
 109 156 127 134 125 116 153 134 100 136 121 131 152 123 153 136 101 155
 140 160 112 110 155 140 105 144 143 154 141 109 102 102 102 117 150 132
 133 149 109 118 134 158 133 102 146 116 147 124 150 143 133 110 105 109
 104 108 115 141 133 101 106 157 134 114 117 143 114 134 158 149 116 130
 136 148 111 102 105 129 135 134 120 159 134 124 133 143 139 118 105 123
 137 114 143 105 117 146 113 137 131 154 117 105 142 145 153 155 101 116
 136 156 156 133 154 137 124 129 135 133 154 154 125 157 112 109 110 130
 102 156 148 101 112 121 110 147 122 155 134 151 149 153 128 132 146 119
 129 135 101 135 127 147 120 152 134 113 145 136 104 126 139 154 100 158
 124 158 135 148 134 137 119 134 143 150 144 140 150 121 139 127 124 125
 125 120 136 126 158 139 114 126 122 146 151 153 138 118 117 150 131 146
 134 142 131 144 134 128 122 105 124 106 159 113]
```

Last, let us evaluate the solution file based on the randomly generated knapsack sizes.

Method1 - Using Dynamic Programming

In [2]:
```
submitted_file_path = "test_knapsack_solution.csv"

#-------------------------------------------------------------------------
# knapsack logic definition, return the table of dynamic programming result: "K"
#-------------------------------------------------------------------------
def knapSack(W, wt, val, n):

    # W = capacity
    # wt = weight
    # val = value
    # n = number of boxes (or stocks,goods, etc... )

    K = [[0 for x in range(W + 1)] for x in range(n + 1)]

    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif wt[i-1] <= w:
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    return K

#-------------------------------------------
# Definition: Dynamic Programming Evaluation
#-------------------------------------------
def Evaluate_Sol(file_path):


    #-------------------------------------------
    # Import value and weight data as list or array
    #-------------------------------------------
    f = open(submitted_file_path, "r")
    rows = [rows for rows in f]

    for i in range(len(rows)):
            rows[i] = rows[i].split(",")

    value = rows[1::3]
    weight = rows[::3]

    value_list = np.float_(value)  # change type of number to interger and float
    weight_list = np.int_(weight)


    #-------------------------------------------------------------
    # Find max value under each capacity and calculate total value
    #-------------------------------------------------------------
    total_element=[]  # store all index of selected element
    result=[]  # store max value of each warehouse instance

    for c in range(len(cap)):
        val = value_list[c]
        wt= weight_list[c]
```

```
        n = len(val)
        W = cap[c]
        x = knapSack(W, wt, val, n)  # x represents the table of knapsack solutio
        result.append(x[n][W])  # x[n][w] is at the bottom right of the table, re


        #------------------------------
        # Find index of selected element
        #------------------------------
        value_left = x[n][W]
        weight_left = W
        element=[]  # store index of selected element of each warehouse instance

        for i in range(n, 0, -1):

            if value_left <= 0 or weight_left <= 0:
                break

            if value_left == x[i - 1][weight_left]:
                continue
            else:
                element.append(i - 1)
                value_left -= val[i - 1]
                weight_left -= wt[i - 1]



        total_element.append(element)

    total_value = np.sum(result)    # calculate total value


    #-------------------------------------------------------------------------
    # Create list of 0 an 1 for selected items and write into test_knapsack_solut
    #-------------------------------------------------------------------------
    final_element = []   # store list of all [0,1]

    for i in total_element:
        total_element_list = []  #store list of [0,1] of each warehouse instance
        for k in range(n):
            if k in i:
                total_element_list.append(1)
            else:
                total_element_list.append(0)
        final_element.append(total_element_list)

    f2 = open(submitted_file_path, "w+")
    write = csv.writer(f2, lineterminator="\n")

    for i in range(len(value_list)):
        write.writerow(weight_list[i])
        write.writerow(value_list[i])
        write.writerow(final_element[i])

    f2.close()
```

```
    current_obj = total_value
    student_id =  "490851"

    return {student_id: current_obj}


final_results = Evaluate_Sol(submitted_file_path)
final_results
```

Out[2]: {'490851': 73795.67}

Conclusion:

The total value of each Warehouse Instance is $73795.67 after using dynamic programming approach.

Method2 - Using Monte-Carlo Simulation

In [3]:
```
#---------------------------------------------
# Import value and weight data as list or array
#---------------------------------------------
f = open(submitted_file_path, "r")
rows = [rows for rows in f]

for i in range(len(rows)):
        rows[i] = rows[i].split(",")

value = rows[1::3]
weight = rows[::3]

value_list = np.float_(value)
weight_list = np.int_(weight)


#-------------------------------------------------
# Find max and min value as the range of simulation
#-------------------------------------------------
max_value = []
min_value = []
for i in value_list:
    max_value.append(max(i))
    min_value.append(min(i))

max_v = max(max_value)
min_v = min(min_value)

#-------------------------------------------------------------------------------
# Definition: Assign 50 random value between maximum and minimum value from origi
#-------------------------------------------------------------------------------
def random_value(Min, Max, n):
    value = np.random.uniform(Min, Max, n)
    value= np.around(value,2)

    return value



#-----------------------------------
# Definition: Monte-Carlo Simulation
#-----------------------------------
def monte_carlo_evaluation(N):
    value_list_mc = []    # store the value for simulation result
    mean_value_mc = []    # store the mean value for all simulation result


    for c in range(len(cap)):
        for trial in range(N):
            W = cap[c]
            wt= weight_list[c]
            n = len(wt)
            Each_value = random_value(min_v, max_v, n) # generate random value fo
            val = Each_value

            x_mc = knapSack(W, wt, val, n)   # x_mc represents the table of knaps
```

```
            value_list_mc.append(x_mc[n][W]) #x_mc[n][w] is at the bottom right c

        mean_value_mc.append(np.mean(value_list_mc))

    total_value_mc = np.around(np.sum(mean_value_mc),2) # Total value of each war

    student_id = "490851"
    current_obj = total_value_mc


    return {student_id: current_obj}

final_results_mc =  monte_carlo_evaluation(1000)   # Each warehouse instance simu
final_results_mc
```

Out[3]:
```
{'490851': 533529.45}
```

Conclusion

The total value of each Warehouse Instance is 533529.45 . The number is far from the total value in previous approach, which is 73795.67, but it is explainable: The mean of randomly assigned value is larger than the mean of original value. Based on the calculation below, the mean of original dataset is 63.17, while the mean of randomly assigned dataset is nearly 160.

In [23]:
```
#-------------------------
# Mean of Original dataset
#-------------------------
mean_list=[]
for i in value_list:
    mean_list.append(np.mean(i))
total_mean_original = np.mean(np.mean(mean_list))
total_mean_original


#-------------------------------
#Mean of random assigned dataset
#-------------------------------
mean_list_mc=[]

for c in range(50):
    n=50
    Each_value = random_value(min_v, max_v, n)
    mean_list_mc.append(np.mean(Each_value))
total_mean_mc = np.mean(mean_list_mc)
print("The mean of orginal dataset is:",total_mean_original, "The mean of random
```

```
The mean of orginal dataset is: 63.11771533333334 The mean of random assigned d
ataset is: 164.563436
```