The code implements linear regression with one variable to predict profits for a food truck. Suppose you are the CEO of a restaurant franchise and are considering different cities for opening a new outlet. The chain already has trucks in various cities and you have data for profits and populations from the cities. You would like to use this data to help you select which city to expand to next.

The file ex1data1.txt contains the dataset for our linear regression problem. The first column is the population of a city (in 10,000s) and the second column is the profit of a food truck in that city (in $10,000s). A negative value for profit indicates a loss.

[**ex1data1.txt**](https://mylearning.suny.edu/content/enforced/471760-2023SP-NEW-EGC493-944/csfiles/home_dir/courses/spring22_merged_egc493_ege593_fouda/ex1data1.txt?_&d2lSessionVal=Ad2jVq8QSl3BSRylDSuWQKrKm)

**Computing the cost J(θ)**

As you perform gradient descent to learn to minimize the cost function J(θ), it is helpful to monitor the convergence by computing the cost. In this section, you will implement a function to calculate J(θ) so you can check the convergence of your gradient descent implementation.

Your next task is to complete the code for the function computeCost which computes J(θ). As you are doing this, remember that the variables X and y are not scalar values. X is a matrix whose rows represent the examples from the training set and y is a vector whose each element represent the value at a given row of X.

Hint: you might need to use the dot() function

**Code:**

# used for manipulating directory paths  
import os

# Scientific and vector computation for python  
import numpy as np

# Plotting library  
from matplotlib import pyplot  
from mpl\_toolkits.mplot3d import Axes3D  # needed to plot 3-D surfaces  
import seaborn as sns  
sns.set()

# Read comma separated data  
data = np.loadtxt(os.path.join('../input/ex1data1', 'ex1data1.txt'), delimiter=',')  
X, y = data[:, 0], data[:, 1]

m = y.size  # number of training examples

    # Add a column of ones to X. The numpy function stack joins arrays along a given axis.   
# The first axis (axis=0) refers to rows (training examples)   
# and second axis (axis=1) refers to columns (features).  
X = np.stack([np.ones(m), X], axis=1)

def computeCost(X, y, theta):     
    # initialize some useful values  
    m = y.size  # number of training examples  
      
    # You need to return the following variables correctly  
    J = 0  
      
    # ====================== YOUR CODE HERE =====================  
      
  
      
    # ===========================================================  
    return J

J = computeCost(X, y, theta=np.array([0.0, 0.0]))  
print('With theta = [0, 0] \nCost computed = %.2f' % J)  
print('Expected cost value (approximately) 32.07\n')

# further testing of the cost function  
J = computeCost(X, y, theta=np.array([-1, 2]))  
print('With theta = [-1, 2]\nCost computed = %.2f' % J)  
print('Expected cost value (approximately) 54.24')