Restarted .venv

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


from mpl_toolkits.mplot3d import axes3d
from warmUpExercise import warmUpExercise
from computeCost import computeCost
from gradientDescent import gradientDescent
from plotData import plotData
```

In [ ]:
```
#  Instructions
#  ------------
#
#  This file contains code that helps you get started on the
#  linear exercise. You will need to complete the following modules
#  in this exericse:
#
#     warmUpExercise.py
#     plotData.py
#     gradientDescent.py
#     computeCost.py
#     gradientDescentMulti.py
#     computeCostMulti.py
#     featureNormalize.py
#     normalEqn.py
#
#  For this exercise, you will not need to change any code in this file,
#  or any other files other than those mentioned above.
#
# x refers to the population size in 10,000s
# y refers to the profit in $10,000s
```

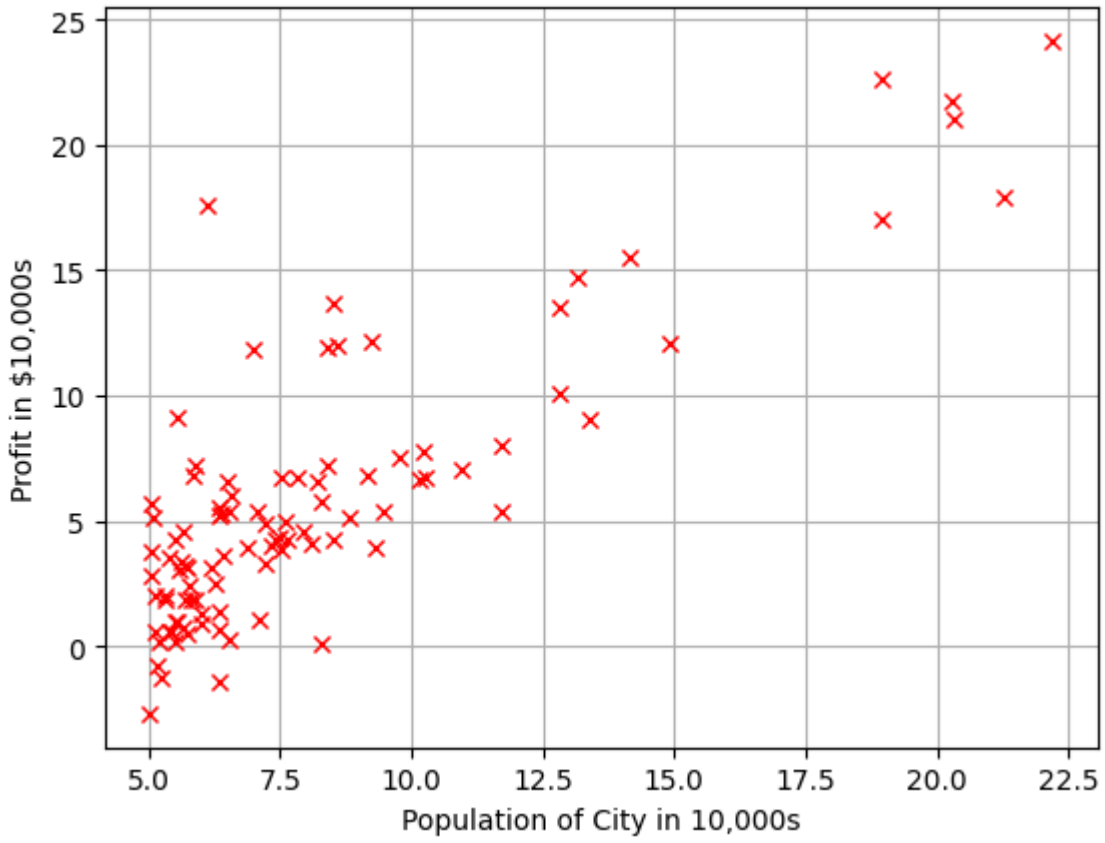
In [ ]:
```
# Complete warmUpExercise.py
print('\n ------------------------- \n')
print('Running warmUpExercise ...')
print('5x5 Identity Matrix:')
warmup = warmUpExercise()
print(warmup)
```

```
-------------------------

Running warmUpExercise ...
5x5 Identity Matrix:
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

In [ ]:
```
# Read data using pandas
path = 'ex1data1.txt'
data = pd.read_csv(path, header=None, names=['Population', 'Profit'])
data.head()

# Résumé des données
```

```
data.describe()



# set X (training data) and y (target variable)
nbCol = data.shape[1]
X = data.iloc[:,0:nbCol-1]
y = data.iloc[:,nbCol-1:nbCol]

# convert from data frames to numpy arrays
X = np.array(X.values)
y = np.array(y.values)

# Plot Data
# Note: You have to complete the code in plotData.py
plotData(X,y)
```

```
/Users/tanguyrdt/Documents/ENSTA_depot/depot-ensta-python/machine_learnin
g/tp1/code/plotData.py:26: UserWarning: FigureCanvasAgg is non-interactiv
e, and thus cannot be shown
  fig.show()
```

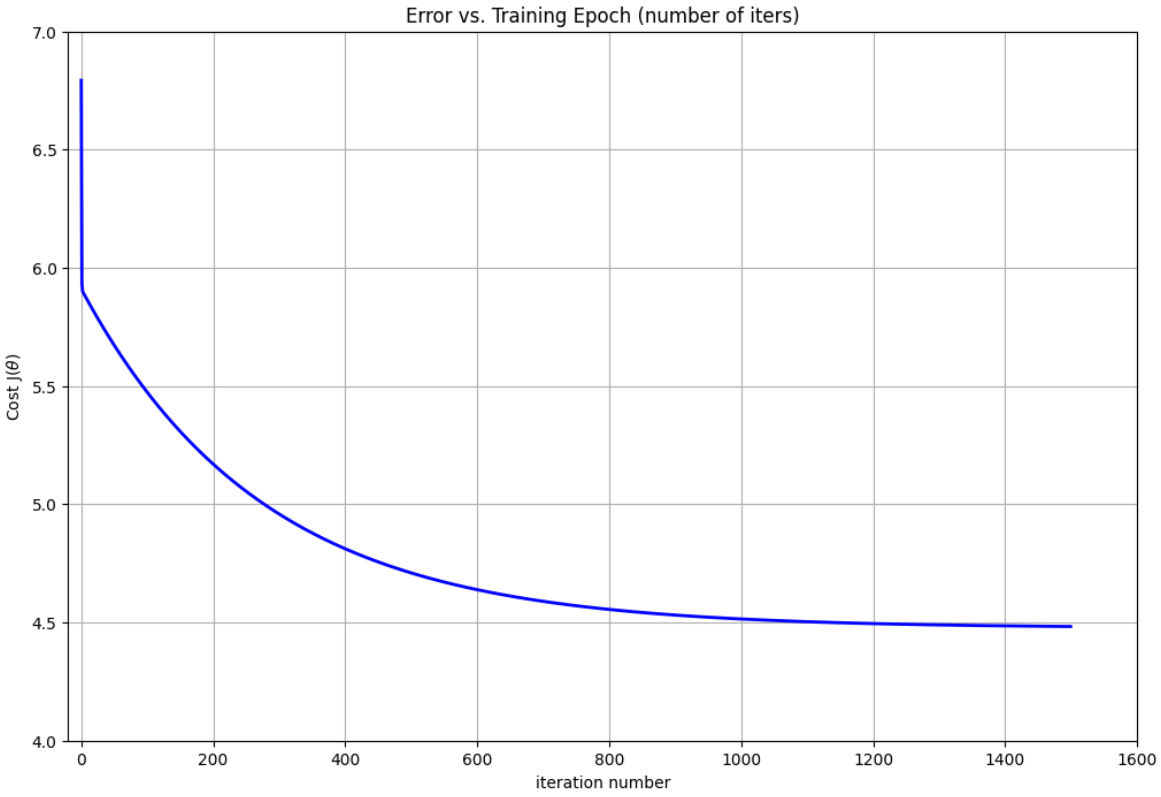


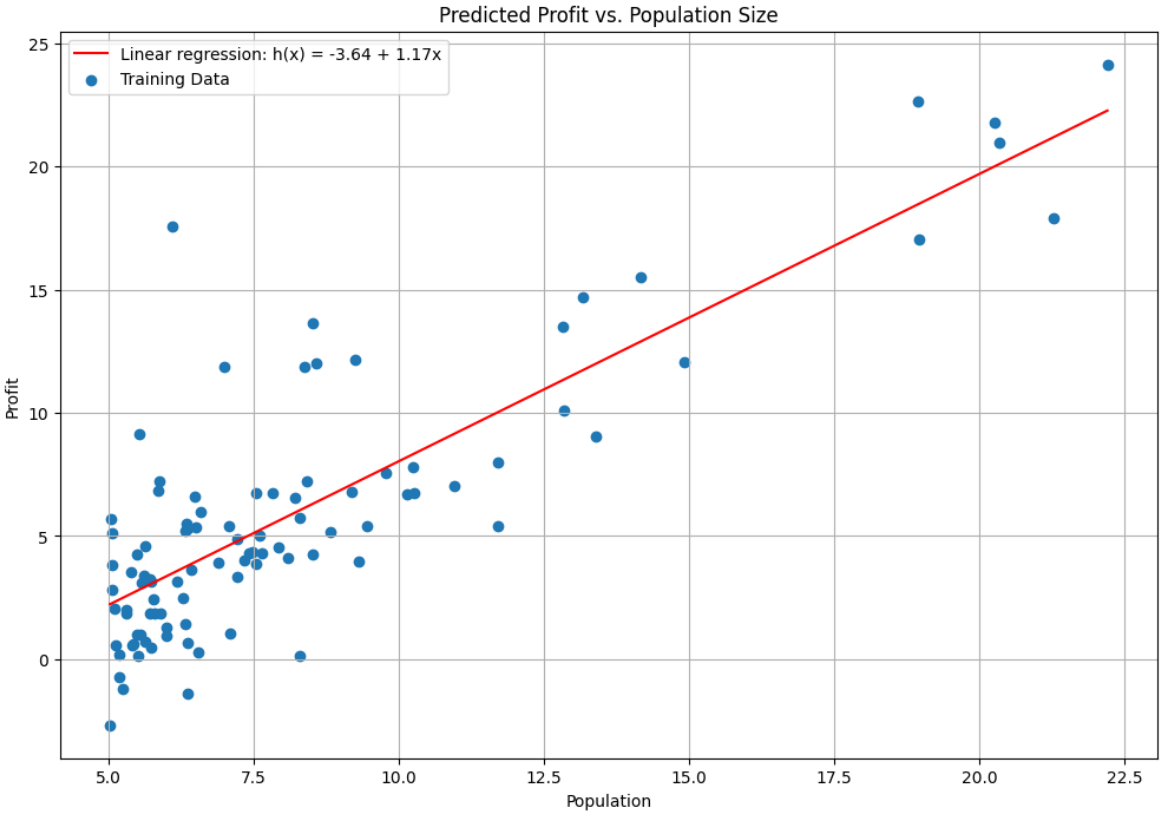
In [ ]:
```
m = X.shape[0]

# Add intercept term to X
#X = np.concatenate((np.ones((m, 1)), X), axis=1)
X = np.column_stack((np.ones((m, 1)), X)) #works fine too

# initialize theta
theta = np.array([[0.,0.]]).T

# # compute and display initial cost
# # Note: You have to complete the code in computeCost.py
J = computeCost(X, y, theta)
```

```
print('\n ------------------------- \n')
print('cost: %0.4f ' % J)
print('Expected cost value (approx) 32.07')

# # further testing of the cost function
J = computeCost(X, y, np.array([[-1, 2]]).T)
print('\n ------------------------- \n')
print('With theta = [-1 ; 2] Cost computed = %f' %J)
print('Expected cost value (approx) 54.24')




# # compute Descent gradient
# # initialize variables for learning rate and iterations
alpha = 0.01
iters = 1500

# # perform gradient descent to "fit" the model parameters
# # Note: You have to complete the code in gradientDescent.py
theta, cost_history, theta_history = gradientDescent(X, y, theta, alpha,


# # print theta to screen
print('\n ------------------------- \n')
print('Theta found by gradient descent: ')
print('%s %s' % (theta[0,0], theta[1,0]))
print('Expected theta values (approx)')
print(' -3.6303  1.1664')


# # Checking the convergence
# # Evolution du coût
fig= plt.figure(figsize=(12,8))
ax = plt.gca()
ax.plot(np.arange(iters), cost_history, color="blue", linewidth=2.0, line
ax.set_xlabel('iteration number')
ax.set_ylabel(r'Cost J($\theta$)')
ax.set_title('Error vs. Training Epoch (number of iters)')
ax.grid()
ax.set_xlim([-20,1600])
ax.set_ylim([4,7])


# # Checking the goodness-of-fit
# # Fit: calcul de la droite de régression
x = np.linspace(data.Population.min(), data.Population.max(), 100)
f = theta[0, 0] + (theta[1, 0] * x)


# # Plot the linear fit
fig = plt.figure(figsize=(12,8))
ax = plt.gca()
ax.plot(x, f, 'r', label='Linear regression: h(x) = %0.2f + %0.2fx'%(thet
ax.scatter(data.Population, data.Profit, label='Training Data')
ax.legend(loc=2)
ax.set_xlabel('Population')
ax.set_ylabel('Profit')
ax.set_title('Predicted Profit vs. Population Size')
ax.grid()
```

```
fig.show()



# # Predict values for population sizes of 35,000 and 70,000
predict1 = np.array([[1, 3.5]]).dot(theta)
predict2 = np.array([[1, 7]]).dot(theta)
#predict1 = np.array([[1, 3.5]])@theta
#predict2 = np.array([[1, 7]])@theta

print('\n -------------------------- \n')
print('For population = 35,000, we predict a profit of {:.4f}'.format(pre
print('For population = 70,000, we predict a profit of {:.4f}'.format(pre
```

```
 --------------------------

cost: 32.0727
Expected cost value (approx) 32.07

 --------------------------

With theta = [-1 ; 2] Cost computed = 54.242455
Expected cost value (approx) 54.24

 --------------------------

Theta found by gradient descent:
-3.6360634754795016 1.1669891581648786
Expected theta values (approx)
 -3.6303  1.1664

 --------------------------

For population = 35,000, we predict a profit of 4483.9858
For population = 70,000, we predict a profit of 45328.6063
<ipython-input-5-57c01103b591>:15: DeprecationWarning: Conversion of an ar
ray with ndim > 0 to a scalar is deprecated, and will error in future. Ens
ure you extract a single element from your array before performing this op
eration. (Deprecated NumPy 1.25.)
  print('cost: %0.4f ' % J)
<ipython-input-5-57c01103b591>:21: DeprecationWarning: Conversion of an ar
ray with ndim > 0 to a scalar is deprecated, and will error in future. Ens
ure you extract a single element from your array before performing this op
eration. (Deprecated NumPy 1.25.)
  print('With theta = [-1 ; 2] Cost computed = %f' %J)
<ipython-input-5-57c01103b591>:74: UserWarning: FigureCanvasAgg is non-int
eractive, and thus cannot be shown
  fig.show()
```





In [ ]:
```
print('\n ------------------------- \n')
print('Visualizing J(theta_0, theta_1) ...')

# Create grid coordinates for plotting
theta0 = np.linspace(-10, 10, 100)
theta1 = np.linspace(-1, 4, 100)
theta0, theta1 = np.meshgrid(theta0, theta1, indexing='xy')

Z = np.zeros((theta0.shape[0],theta1.shape[0]))

# Calculate Z-values (Cost) based on grid of coefficients
```

```
for (i,j),v in np.ndenumerate(Z):
    t = np.array([[theta0[i,j], theta1[i,j]]]).T
    Z[i,j] = computeCost(X,y, t)


fig = plt.figure(figsize=(15,6))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122, projection='3d')

# Left plot
CS = ax1.contour(theta0, theta1, Z, np.geomspace(Z.min(),Z.max(),10), cma
plt.clabel(CS, inline=1, fontsize=10)
ax1.scatter(theta_history[0,:],theta_history[1,:], c='r')
ax1.grid()

# # Right plot
ax2.plot_surface(theta0, theta1, Z, rstride=1, cstride=1, alpha=0.6, cmap
ax2.set_zlabel('Cost')
ax2.set_zlim(Z.min(),Z.max())
ax2.view_init(elev=15, azim=230)
ax2.grid()

# settings common to both plots
for ax in fig.axes:
    ax.set_xlabel(r'$\theta_0$', fontsize=17)
    ax.set_ylabel(r'$\theta_1$', fontsize=17)
```

```
Visualizing J(theta_0, theta_1) ...
```

```
<ipython-input-6-4cd7af9e400b>:15: DeprecationWarning: Conversion of an ar
ray with ndim > 0 to a scalar is deprecated, and will error in future. Ens
ure you extract a single element from your array before performing this op
eration. (Deprecated NumPy 1.25.)
  Z[i,j] = computeCost(X,y, t)
<ipython-input-6-4cd7af9e400b>:23: UserWarning: The following kwargs were
not used by contour: 'color'
  CS = ax1.contour(theta0, theta1, Z, np.geomspace(Z.min(),Z.max(),10), cm
ap=plt.cm.jet, color='black')
```