# Simpe Linear Regression

```
import numpy as np
import matplotlib.pyplot as plt
import random
import seaborn as sns
from tqdm import tnrange, tqdm_notebook
from IPython.display import HTML, display
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use
  import pandas.util.testing as tm
```

In this workshop we will use **linear regression** to predict data

We start by create a data from a linear equation y = mx + c as seen below.

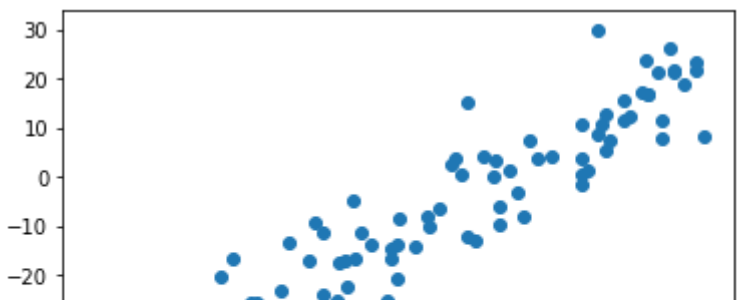
In reality, the data should come from some experiment or real world data collection. For now, we will stick with random data.

```
# Initialize m=7 and c=-10
m = 7
c = -10
```

```
# create an empty list named X_list and Y_list for values of x and y, respectively
X_list = []
Y_list = []

# P_tuple is a list for a tuple of (x,y)
P_tuple = []

# loop with i from 0 to 99, inclusive
for i in range(0, 100):

  # assign a random value to x (in the range of -5 to 5)
  x = random.uniform(-5,5)
  X_list.append(x)

  # Create a y value for the generated x, we also add a "noise" to X
  # This "noise" is added to illustrate example from real world
  rd = random.random()* 10 * random.uniform(-2, 2)
  y = (m*x) + c + rd
  Y_list.append(y)

  P_tuple.append((x,y))

  print("Generate x="+str(x)+ ", y="+str(y) +", noise="+str(rd))
```

```
Generate x=-4.143293240130798, y=-34.64529427402296, noise=4.357758406892631
Generate x=0.44313864610872944, y=-8.105135065439644, noise=-1.20710558820075
Generate x=-2.1724875170214197, y=-27.989929558583466, noise=-2.782516939433528
Generate x=-1.7992809245674737, y=-13.444277130963016, noise=9.1506893410093
Generate x=-2.8100317091780957, y=-41.86581268378081, noise=-12.195590719534136
Generate x=-0.19850212639745823, y=-25.273900941159578, noise=-13.88438605637737
Generate x=-4.953723571509348, y=-40.30508927306991, noise=4.370975727495525
Generate x=3.262428054583763, y=10.762611976505433, noise=-2.0743844055809078
Generate x=-0.1319305261130861, y=-16.606713224851553, noise=-5.683199542059953
Generate x=-0.06313751514756838, y=-20.604651621159327, noise=-10.162689015126347
Generate x=-1.482762183026872, y=-17.19694210546966, noise=3.1823931757184467
Generate x=-0.7206336732352936, y=-16.587751779564204, noise=-1.5433160669171486
Generate x=-4.903334569842167, y=-42.4900513074315, noise=1.8332906814636694
Generate x=3.332007969075743, y=5.515216769538843, noise=-7.808839013991359
Generate x=2.9489618390318606, y=10.461475627216199, noise=-0.18125724600682522
Generate x=-2.9880115168889443, y=-50.72128577892876, noise=-19.805205160706148
Generate x=-4.155568725334847, y=-42.779619205842096, noise=-3.6906381284981675
Generate x=-4.35937164416739, y=-40.81940396940067, noise=-0.3038024602289359
Generate x=-1.9344129263973775, y=-23.014171648747638, noise=0.5267188360340052
Generate x=1.0827771200303307, y=15.25063074372441, noise=17.671190903512095
Generate x=-4.977747655891424, y=-50.23143714140826, noise=-5.387203550168285
Generate x=4.027945653748741, y=16.867357164871915, noise=-1.328262411369273
Generate x=3.9828546179998305, y=23.82201952887828, noise=5.942037202879467
Generate x=-0.043672210872360395, y=-13.779393205465544, noise=-3.4736877293590207
Generate x=-1.2453404754285389, y=-11.300924149139083, noise=7.4164591788606895
Generate x=-1.0215466282649, y=-25.330370400063167, noise=-8.179544002208868
Generate x=4.370162814020322, y=26.28070950960123, noise=5.689569811458978
Generate x=3.7152161469025256, y=12.432616009143857, noise=-3.5738970191738204
Generate x=-3.698810423122312, y=-37.87226606818507, noise=-1.980593106328883
Generate x=1.4997989953370103, y=0.18439824303737373, noise=-0.3141947243216995
Generate x=-4.819077912472572, y=-47.31286188687346, noise=-3.5793164995654556
Generate x=3.936891914962793, y=17.08594588421019, noise=-0.4722975205293591
Generate x=2.0046549207789823, y=-8.137285924557334, noise=-12.16987037001021
Generate x=3.621660961971614, y=11.375763937810312, noise=-3.9758627959909845
Generate x=-3.0135711330537687, y=-31.283271797900348, noise=-0.18827386652396613
Generate x=-0.9033672389443943, y=-17.25410555171086, noise=-0.9305348791001007
Generate x=-1.1534919837598476, y=-31.666724509351376, noise=-13.592280623032444
Generate x=3.194435404431511, y=8.787501428890005, noise=-3.573546402130573
Generate x=-1.234915600375209, y=-24.037182207138457, noise=-5.392773004511995
Generate x=-3.37942477942219, y=-34.19583171274237, noise=-0.5398582567870384
Generate x=-2.9187067408624454, y=-20.257179618668914, noise=10.173767567368204
Generate x=3.330908004361854, y=12.675422385074372, noise=-0.6409336454586081
Generate x=2.089145244281294, y=7.219980450312282, noise=2.5959637403432234
Generate x=-2.432232324372791, y=-25.566749267072062, noise=1.4588770035374745
Generate x=-3.7839404562045975, y=-39.781993684416776, noise=-3.29441049098459
Generate x=-2.458479315343766, y=-36.667816551570496, noise=-9.458461344164133
Generate x=0.6272208325391198, y=-6.535701200816281, noise=-0.9262470285901194
Generate x=-0.021202340800745567, y=-8.385999987512635, noise=1.7624163980925849
Generate x=0.2542167487795659, y=-14.185107030800399, noise=-5.964624272257361
Generate x=4.922712981272134, y=8.2212106446636, noise=-16.23778022424134
Generate x=4.015347110991982, y=16.591617805758382, noise=-1.5158119711854936
Generate x=4.792093092956012, y=23.192567867993958, noise=-0.35208378269812934
Generate x=-2.712160019843627, y=-16.509056612531708, noise=12.476063526373684
Generate x=1.8909904061870186, y=-3.026347468367365, noise=-6.263280311676495
Generate x=-4.88233028383539, y=-43.72880986697135, noise=0.44750211987637967
Generate x=-4.311517324761836, y=-39.506080142583656, noise=0.6745411307491957
Generate x=4.242578003857464, y=11.380276659545965, noise=-8.31776936745628
Generate x=-0.4640085470855162, y=-13.633880104514954, noise=-0.3858202749163409
Generate x=1.55234785068083, y=3.1284271964953794, noise=2.26199224172957
Generate x=4.247428827112966, y=7.8285688481576035, noise=-11.903432941633158
Generate x=-0.1546980117061194, y=-14.79423055429281, noise=-3.7113444723499738
Generate x=-3.7946192705004345, y=-38.34145909940006, noise=-1.7791242058970211
Generate x=3.3949036349727066, y=7.326218868655543, noise=-6.4381065761534035
Generate x=4.609572813307691, y=18.625222361415492, noise=-3.6417873317383447
Generate x=3.057232470045088, y=1.4077276058854853, noise=-9.992899684430133
Generate x=0.9827417015744713, y=0.3017298469537786, noise=3.4225379359324797
Generate x=-3.5234479850532576, y=-33.72423489344625, noise=0.9399010019265537
Generate x=2.451011109571958, y=4.292977439770281, noise=-2.8641003272334253
```

```
# Plotting (x,y) in graphical format

# If we need a line, change to "line=True"
def plot_graph(X_list,Y_list,m,c,linspace=[-5,5],n_data=100,line=False):
  plt.scatter(X_list, Y_list)
  x = np.linspace(linspace[0],linspace[1],n_data)
  y = m*x+c
  if line: plt.plot(x, y, '-r')
  plt.show()

plot_graph(X_list,Y_list,m,c)
```




We can see that the line has a "linear" pattern. This is because it is intrinsically generated from y = 7x + -10 which is a linear equation

Our goal of this lab is to let a computer (our model) learn the behavior of this data and use the learned behavior to "predict" data outside our range.

We will do so by "designing a model". In this case, it will be a linear model. A model is actually realized, we just pick a form (linear model) and we will let the computer realized the actual model by learning.

A linear model is a linear equation of y = mx + c. The computer will try to calculate the value of m and c that "match" the given data as good as possible.

## Hypothesis Function

Formally, our model is called "Hypothesis Function" in ML lingo.

1. Our linear Hypothesis Function is in the following form

> $h(\Theta) = \Theta_1 x + \Theta_0$

We can see that $h(\Theta)$ is actually y

$\Theta_1$ is m

$\Theta_0$ is c

The goal is to find $\Theta_1$ and $\Theta_1$ by computer.

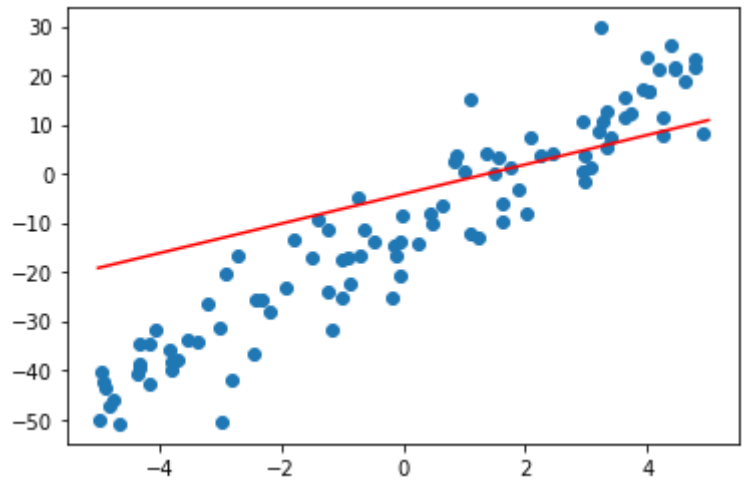
```
# TODO:1-1 Write a python function that return y of our hypothesis function

def hypothesis(theta0, theta1, x):

  return theta1*x+theta0



### Test code for TODO:1 ###

test1 = hypothesis(4.7268, -3.6136, 8.3689)
print(test1)

#####################################
### It should be -25.51505704 ###
#####################################
```

```
-25.51505704
```

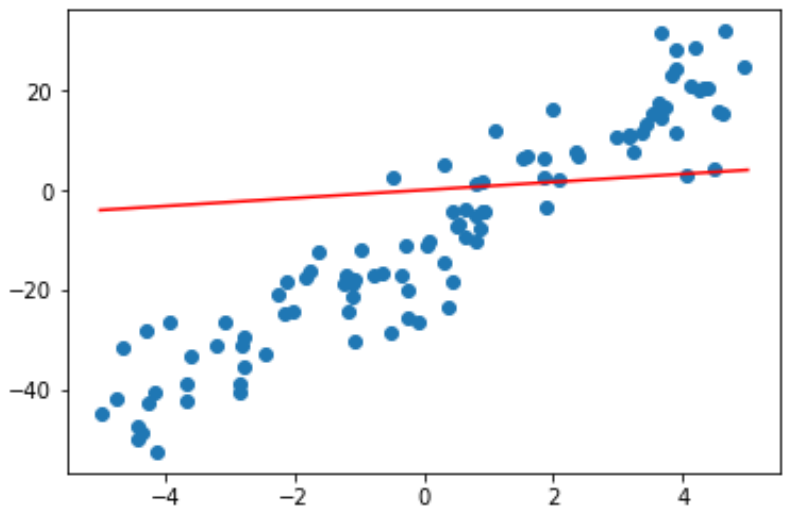
2. Next, we assign initial value for $\Theta_1$ (the variable $m$) and $\Theta_0$ (the variable $c$). The initial value is randomly assigned so that our hypothesis has a starting point.

```
# TODO:1-2 assign "starting" value for theta0 และ theta1, randomly. (you can use any random function BUT NOT FIXED value)

### CODE HERE ######################

theta0 = random.uniform(-5, 5)
theta1 = random.uniform(-5, 5)

#####################################

print("Theta 0 is :",theta0)
print("Theta 1 is :",theta1)

plot_graph(X_list,Y_list,theta1,theta0,line=True)

################################################
### Your result must look like the figure below #
### line and dot might deviate (due to randomness)##
################################################
```

```
Theta 0 is : -4.170078439223048
Theta 1 is : 3.0090925421248276
```



Example of the result. (Your result might varies because of randomness)



The objective here is to show that with simple random, we can "realized" our hypothesis function. It might not be good but it's something.

## Loss Function

Our goal is to find $\Theta_1$ and $\Theta_0$ that is good. So, we have to define what "goodness" is. We create something call **loss function** (sometime called **cost function**). This function calculate how **bad** our current model is. We want to **minimize** the output of the loss function.

Intuitively, a loss function calculate how much our model (the redline) differs from the actual data (the blue dots) over all given data. The further the blue dot from the line, the higher the loss function value.

There are many different loss function. Which one is suitable really depends on the actual problem. For our simple linear regression, we usually use MSE (Mean square error) which can be described by

> MSE = $\frac{1}{n}\sum_{t=1}^{n}(y-\tilde{y})^2$

Where $y$ is the vaue from the actual data, $\tilde{y}$ is the value predicted by our model and $n$ is the number of data point

```
# TODO:2-1 write a loss function

def cal_MSE(y, y_pred):

  ### CODE HERE ##############

  mse = 0
  for i in range(len(y)):
    mse += (y[i]-y_pred[i])**2

  return (mse * 1.0) / len(y)

  ############################


### Test code for TODO:2-1 ###

y3 = [1.50, -2.73, 8.62, -12.14, 4.78]
y3_pred = [1.73, 1.68, -3.57, -10.92, 3.68]

td3 = cal_MSE(y3,y3_pred)
print(td3)

######################################
### The result should be 34.1591 ####
######################################
```

```
########################################
```

```
34.1591
```

Next, we will use our 2-1 to calculate MSE from our X_list and Y_list generated earlier

```
# TODO:2-2 Calculate loss from X_list and Y_list

total_loss = 0

### CODE HERE ########################################

for x,y in zip(X_list,Y_list):
  # calculate this loss
  y_pred = hypothesis(theta0, theta1, x)
  loss = (y-y_pred)**2
  # sum total lost
  total_loss += loss

  print("x=",x,"y=",y,"loss=",loss)

####################################################

print("\nAverage loss =", total_loss/len(X_list))
```

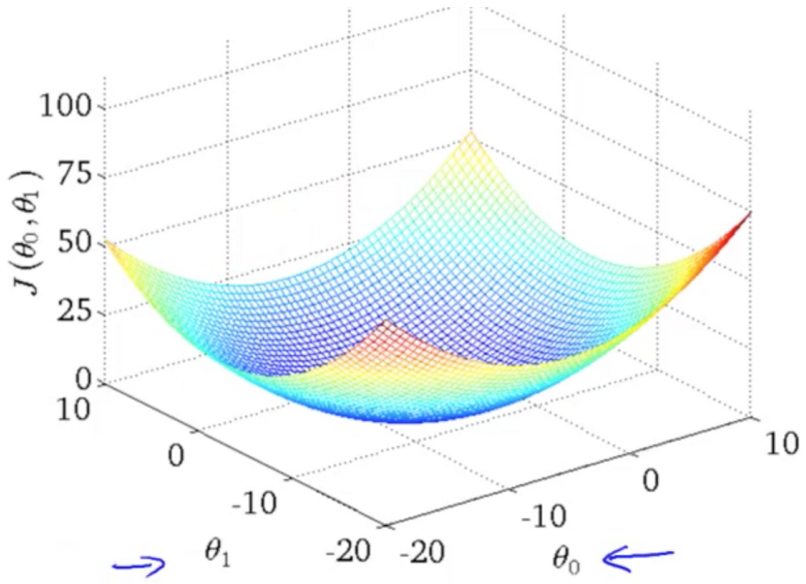
```
x= -4.143293240130798 y= -34.64529427402296 loss= 324.2759283813755
x= 0.44313864610872944 y= -8.105135065439644 loss= 27.75711144156475
x= -2.1724875170214197 y= -27.989929558583466 loss= 298.68947717607654
x= -1.7992809245674737 y= -13.444277130963016 loss= 14.89956819698689
x= -2.8100317091780957 y= -41.86581268378081 loss= 854.982792174313
x= -0.19850212639745823 y= -25.273900941159578 loss= 420.51700298188433
x= -4.953723571509348 y= -40.30508927306991 loss= 450.66187212769074
x= 3.262428054583763 y= 10.762611976505433 loss= 26.17082119802051
x= -0.1319305261130861 y= -16.606713224851553 loss= 144.9530185790304
x= -0.06313751514756838 y= -20.604651621159327 loss= 263.88659237464293
x= -1.482762183026872 y= -17.19694210546966 loss= 73.36085303662203
x= -0.7206336732352936 y= -16.587751779564204 loss= 105.04650914491008
x= -4.903334569842167 y= -42.4900513074315 loss= 555.3273882282679
x= 3.332007969075743 y= 5.515216769538843 loss= 0.11629813334705395
x= 2.9489618390318606 y= 10.461475627216199 loss= 33.152894081235814
x= -2.9880115168889443 y= -50.72128577892876 loss= 1410.7539131341389
x= -4.155568725334847 y= -42.779619205842096 loss= 681.4736306431381
x= -4.35937164416739 y= -40.81940396940067 loss= 553.7349197294924
x= -1.9344129263973775 y= -23.014171648747638 loss= 169.60544947488634
x= 1.0827771200303307 y= 15.25063074372441 loss= 261.227460895591
x= -4.977747655891424 y= -50.23143714140826 loss= 966.1438969740349
x= 4.027945653748741 y= 16.867357164871915 loss= 79.51243204570136
x= 3.9828546179998305 y= 23.82201952887828 loss= 256.2342884828075
x= -0.043672210872360395 y= -13.779393205465544 loss= 89.83060816589456
x= -1.2453404754285389 y= -11.300924149139083 loss= 11.44807883360057
x= -1.0215466282649 y= -25.330370400063167 loss= 327.1165490055465
x= 4.370162814020322 y= 26.28070950960123 loss= 299.3095014757074
x= 3.7152161469025256 y= 12.432616009143857 loss= 29.41180595386075
x= -3.698810423122312 y= -37.87226606818507 loss= 509.5008166329484
x= 1.4997989953370103 y= 0.18439824303737373 loss= 0.02514041398841493
x= -4.819077912472572 y= -47.31286188687346 loss= 820.3488143262448
x= 3.936891914962793 y= 17.08594588421019 loss= 88.53967303674638
x= 2.0046549207789823 y= -8.137285924557334 loss= 99.98799350009493
x= 3.621660961971614 y= 11.375763937810312 loss= 21.603247579658785
x= -3.0135711330537687 y= -31.283271797900348 loss= 325.6248738440962
x= -0.9033672389443943 y= -17.25410555171086 loss= 107.44797471424182
x= -1.1534919837598476 y= -31.666724509351376 loss= 577.2333928932663
x= 3.194435404431511 y= 8.787501428890005 loss= 11.190551150342873
x= -1.234915600375209 y= -24.037182207138457 loss= 260.85895003631765
x= -3.37942477942219 y= -34.19583171274237 loss= 394.29057509263873
x= -2.9187067408624454 y= -20.257179618668914 loss= 53.3548801316043
x= 3.330908004361854 y= 12.675422385074372 loss= 46.546375119844924
```

## Calculation of Derivatives

Now, we have a loss function that we can use to measure our *goodness*. The next step is to gradually adjust our $\theta_0$ and $\theta_1$ in our hypothesis function, hoping that it will *reduce* our loss.

This can be done simply by recalling your basic calculus. Let say we have a real value function $f(x)$. We know that the derivative of $f(x)$ ( $f'(x)$) give us the direction that we should adjust x to minimize $f(x)$.

For our current situation we have a loss function that we want to minimize and the value of our loss function depends on our parameters. So, we calculate a *partial derivative* of the loss function. Calculate one derivative with respect to $\theta_0$ and another one with respect to $\theta_1$.




Please recall your calculus and write a function that calculate partial derivative with resepect to $\theta_0$ and $\theta_1$.

Also recall that our loss function can be written mathematically as.

> $\frac{1}{n}\sum_{t=1}^{n}(y-\tilde{y})^2$

when replacing $\tilde{y}$ with our hypothesis. This result in

> $\frac{1}{n}\sum_{t=1}^{n}(y-(\theta_1 x+\theta_0))^2$

This is the function that you have to calculate partial derivative.

```
x= 1.358781767340984 y= 4.252120759869031 loss= 18.77921459366946
```

```
# TODO:3 Calculate derivatives of the loss function
```

```
def derivatives(theta0, theta1, X_list, Y_list):
  dtheta0 = 0
  dtheta1 = 0

  ### CODE HERE ########################################

  for (xi, yi) in zip(X_list, Y_list):
    dtheta0 += 2*(yi-theta1*xi-theta0)*(-1)
    dtheta1 += 2*(yi-theta1*xi-theta0)*(-xi)

  #might need to finally do something here
  dtheta0 = dtheta0/len(X_list)
  dtheta1 = dtheta1/len(X_list)

  ######################################################

  return dtheta0, dtheta1


### Test code for TODO:3 ###

dt30 , dt31 = derivatives(3.24, 5.66, [1.45, -2.45, 5.38, -5.49], [10.46, -4.25, -5.83, 13.87])

print(dt30, dt31)

#####################################################
### should be -1.89315 และ 114.6570925 ####### (wrong ans.)
#####################################################
```

```
-3.7863000000000007 229.314185
```

The gradient value give us the *direction* that we should adjust our $\theta_0$ and $\theta_1$. But we don't really know how much should we adjust on that direction. Here comes another parameter in linear regression, **learning rate** (sometime called **alpha**). This value is use to adjust how far should we move along that direction. As usual, the best value of learning rate is different for each problem but we should start small, 0.1 is a good place to start.

The next task is to write a function that adjust our model parameters ($\theta_0$ and $\theta_1$) along the gradient direction using learning rate.

```
# TODO:4 Adjusting model paremter (theta1 and theta0)

def updateParameters(theta0, theta1, X_list, Y_list, alpha):

  ### CODE HERE ########################################################

  # Step1: Calculate derivative
  dtheta0, dtheta1 = derivatives(theta0, theta1, X_list, Y_list)

  # Step2: adjust theta1 and theta 0 along gradient (dtheta0, dtheta1) by alpha
  theta0 -= alpha*dtheta0
  theta1 -= alpha*dtheta1

  ######################################################################

  return theta0, theta1


### Test code for TODO:4 ###

tt40, tt41 = updateParameters(3.24, 5.66, [1.45, -2.45, 5.38, -5.49], [10.46, -4.25, -5.83, 13.87], 0.1)

print(tt40,tt41)

#########################################################################
### should be 3.4293150000000003 และ -5.805709250000001 ####### (wrong ans.)
#########################################################################
```

```
2.86137 28.591418500000003
```

We will repeatedly call updateParameters until we believe that the model closely match the data. For now, we will just do this for **epoch** round.

Finally, we can combine everything to create a linear regression algorithm.

```
# TODO:5 Write a LinearRegression algorithm
# Input: X_list and Y_List
# Output: theta1 and theta0 that gives a model that (hopefully) best match X_List and Y_List
```

```
def LinearRegression(X_list, Y_list, epoch, linspace=[-5,5], n_data=100,alpha=0.0002):
  ### CODE HERE ################################################################

  # Step1: randomly assign intiial value for theta1 and theta0
  theta0 = random.uniform(-5, 5)
  theta1 = random.uniform(-5, 5)

  # Step2: Repeat for epoch times
  for i in range(0, epoch):
    # Step3: To get some visualization, we will plot our data and model every 100 step
    if i % 100 == 0:
      pass
      # plot_graph(X_list,Y_list,theta1,theta0,linspace=linspace, n_data=n_data, line=True)
    # Step4: this is where we actually adjust theta0 and theta1
    theta0, theta1 = updateParameters(theta0, theta1, X_list, Y_list, alpha)

  return theta1,theta0


### Test code for TODO:5 ###
### Please call me #######

new_m, new_c = LinearRegression(X_list, Y_list, epoch=8000, alpha=0.001)
```

Printing new_m and new_c, it should be close to that we set in the very beginning of this notebook

```
print(new_m, new_c)
```

```
6.662576658409021 -11.571163623271687
```

## Let's predict!!!

Now we have our model. We can use our model to predict for *any* value of x.

```
# randomly pick some value from X_list

index = random.randint(0,len(X_list)-1)
x_rand = X_list[index]

# use our hypothesis with the learned parameter to predict Y

pred = hypothesis(new_c, new_m, x_rand)

print("x_rand :",x_rand,"pred :",pred," Y_real :", Y_list[index])
```

```
x_rand : 1.6105502780633305 pred : -0.8407489334527831  Y_real : -6.040189349216186
```

## Real world data

```
# We will download salary dataset Salary_Data.csv [cr.Kaggle] https://www.kaggle.com/kaggle/sf-salaries
# This is real world data set of a salaries in San Francisco

! wget https://raw.githubusercontent.com/Mixelon-tera/Workshop6_LinearRegression/master/Salary_Data.csv
```

```
--2020-08-25 07:49:22--  https://raw.githubusercontent.com/Mixelon-tera/Workshop6_LinearRegression/master/Salary_Data.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.0.133, 151.101.64.133, 151.101.128.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.0.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 454 [text/plain]
Saving to: ‘Salary_Data.csv’

Salary_Data.csv     100%[===================>]     454  --.-KB/s    in 0s

2020-08-25 07:49:22 (19.2 MB/s) - ‘Salary_Data.csv’ saved [454/454]
```

```
# Adjust data from csv format to Year_list and  Salary_list which is "work year" and "salary", respectively

import pandas as pd

Year_list = []
Salary_list = []

df = pd.read_csv("Salary_Data.csv")
```

```
for i in range(len(df)):
  Year_list.append(df['YearsExperience'][i])
  Salary_list.append(df['Salary'][i])


# Using linear regression
my_m, my_c = LinearRegression(Year_list, Salary_list, linspace=[0,12],n_data=len(df),epoch=2000, alpha=0.01)


print(my_m, my_c)
```

```
9450.698199363713 25787.2413127563
```

## Predicting salary from work year

```
# random a year from X_list

my_index = random.randint(0,len(Year_list)-1)
my_x_rand = Year_list[my_index]

# predict Y

my_pred = hypothesis(my_c, my_m, my_x_rand)
print("my_x_rand :",my_x_rand,"my_pred :",my_pred," Salary_real :", Salary_list[my_index])
```

```
my_x_rand : 3.7 my_pred : 60754.824650402035  Salary_real : 57189.0
```