model = LinearRegression()

This statement creates the variable model as the instance of LinearRegression. You can provide several optional parameters to LinearRegression:

* **fit\_intercept** is a Boolean (True by default) that decides whether to calculate the intercept 𝑏₀ (True) or consider it equal to zero (False).
* **normalize** is a Boolean (False by default) that decides whether to normalize the input variables (True) or not (False).
* **copy\_X** is a Boolean (True by default) that decides whether to copy (True) or overwrite the input variables (False).
* **n\_jobs** is an integer or None (default) and represents the number of jobs used in parallel computation. None usually means one job and -1 to use all processors.

#Step 1: Import packages and classes

# import the package numpy and the class LinearRegression from sklearn.linear\_model

#Step 2: Provide data

#defining data to work with. The inputs,x and output,y  should be arrays

# .reshape() on x because this array is required to be two-dimensionalto have one column and as many rows as necessary.

#Step 3: Create a model and fit it

# create a linear regression model and fit it using the existing data

# With .fit(), you calculate the optimal values of the weights 𝑏₀ and 𝑏₁, using

# the existing input and output (x and y) as the arguments.

# In other words, .fit() fits the model. It returns self, which is the variable model itself.

#Step 4: Get results

#Once you have your model fitted, you can get the results to check whether the model works satisfactorily and interpret it.

#You can obtain the coefficient of determination (𝑅²) with .score() called on model

# you get coefficient of determination: 0.715875613747954

# The attributes of model are .intercept\_, which represents the coefficient, 𝑏₀

#and .coef\_, which represents 𝑏₁

# you get intercept: 5.633333333333329

# you get slope: [0.54]

# You can notice that .intercept\_ is a scalar, while .coef\_ is an array

#x\_predict=16

# verify result for x\_predict=x using hard coding

#y\_pred\_verify = model.intercept\_ + model.coef\_ \* x\_predict   # why model.coef\_ ? why not model.coef?

#print('verify response:', y\_pred\_verify, sep='\n')

#plt.axis([0, 60, 0, 50])

import numpy as np

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

x=np.array([5,15,25,35,45,55])

xm=x

x=x.reshape((-1,1))

y=np.array([11,16,18,30,22,38])

ym=y

y.shape

plt.scatter(x,y)

model = LinearRegression()

model

model.fit(x,y)

r\_sq=model.score(x,y)

print("COEFF OF DETERMINATION = ",r\_sq)

intercept=model.intercept\_

print("intercept = ",intercept)

slope=model.coef\_

print("intercept = ",slope)

x\_predict =x

y\_predict = model.predict(x\_predict)

print("actual response:",y,sep="\n")

print("predict response:",y\_predict,sep="\n")

plt.scatter(x,y)

plt.plot(x\_predict,y\_predict,"r-")

plt.xlabel("independent Variable, x")

plt.ylabel("dependent Variable, y")

plt.title("DATA POINTS")

ybar = np.array([np.mean(y)]\*6)

plt.plot(x\_predict,ybar,"g-")

x=2\*np.random.rand(100,1)

y=4 +(3\*x) + (2\*np.random.rand(100,1))

plt.scatter(x,y)

x=2\*np.random.rand(100,1)

y=4 -(3\*x) + (2\*np.random.rand(100,1))

plt.scatter(x,y)

x=2\*np.random.rand(100,1)

y=4 +(3\*x) + (10\*np.random.rand(100,1))

plt.scatter(x,y)

b1 = (np.mean(x)\*np.mean(y) - np.mean(x\*y))/np.mean(x)\*\*2-np.mean(x\*\*2)

b0 = np.mean(y)-b1\*(np.mean(x))

y\_predict1=b0+b1\*x

ssr=np.sum(np.square(y\_predict1-np.mean(y)))

sse=np.sum(np.square(y-y\_predict1))

ssto = np.sum(np.square(y-np.mean(y)))

rs = ssr/ssto

rs

