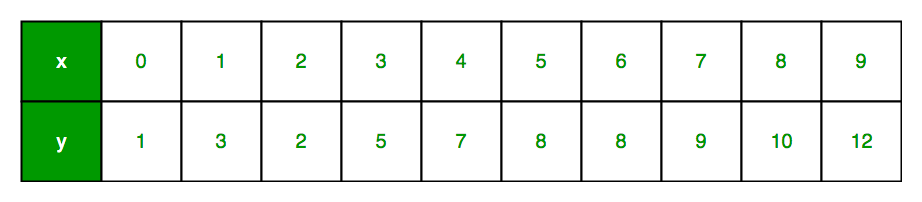
Linear Regression (Python Implementation)

## Simple Linear Regression

Simple linear regression is an approach for predicting a **response** using a **single feature**.

t is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

Let us consider a dataset where we have a value of response y for every feature x:



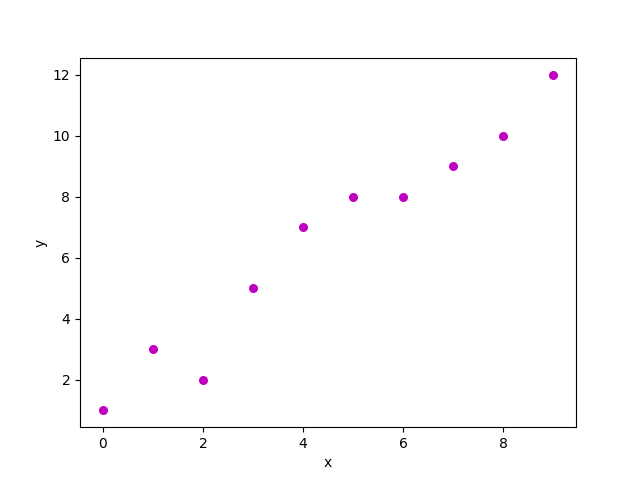
For generality, we define:

x as **feature vector**, i.e x = [x\_1, x\_2, …., x\_n],

y as **response vector**, i.e y = [y\_1, y\_2, …., y\_n]

for **n** observations (in above example, n=10).

A scatter plot of above dataset looks like:-



Now, the task is to find a **line which fits best** in above scatter plot so that we can predict the response for any new feature values. (i.e a value of x not present in dataset)

This line is called **regression line**.

The equation of regression line is represented as:

Here,

* h(x\_i) represents the **predicted response value** for ith observation.
* b\_0 and b\_1 are regression coefficients and represent **y-intercept** and **slope** of regression line respectively.

To create our model, we must “learn” or estimate the values of regression coefficients b\_0 and b\_1. And once we’ve estimated these coefficients, we can use the model to predict responses!

In this article, we are going to use the **Least Squares technique**.

Now consider:

Here, e\_i is **residual error** in ith observation.  
So, our aim is to minimize the total residual error.

We define the squared error or cost function, J as:

and our task is to find the value of b\_0 and b\_1 for which J(b\_0,b\_1) is minimum!

Without going into the mathematical details, we present the result here:

where SS\_xy is the sum of cross-deviations of y and x:

and SS\_xx is the sum of squared deviations of x:

Note: The complete derivation for finding least squares estimates in simple linear regression can be found here.

Given below is the python implementation of above technique on our small dataset:

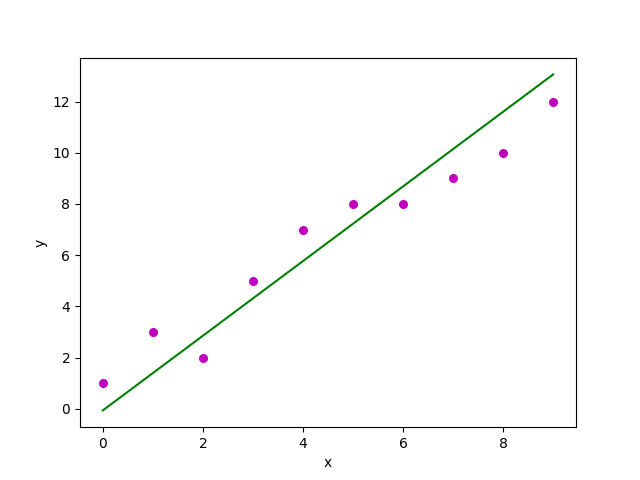
|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt    def estimate\_coef(x, y):      # number of observations/points      n = np.size(x)        # mean of x and y vector      m\_x, m\_y = np.mean(x), np.mean(y)        # calculating cross-deviation and deviation about x      SS\_xy = np.sum(y\*x - n\*m\_y\*m\_x)      SS\_xx = np.sum(x\*x - n\*m\_x\*m\_x)        # calculating regression coefficients      b\_1 = SS\_xy / SS\_xx      b\_0 = m\_y - b\_1\*m\_x        return(b\_0, b\_1)    def plot\_regression\_line(x, y, b):      # plotting the actual points as scatter plot      plt.scatter(x, y, color = "m",                 marker = "o", s = 30)        # predicted response vector      y\_pred = b[0] + b[1]\*x        # plotting the regression line      plt.plot(x, y\_pred, color = "g")        # putting labels      plt.xlabel('x')      plt.ylabel('y')        # function to show plot      plt.show()    def main():      # observations      x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])      y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])        # estimating coefficients      b = estimate\_coef(x, y)      print("Estimated coefficients:\nb\_0 = {}  \            \nb\_1 = {}".format(b[0], b[1]))        # plotting regression line      plot\_regression\_line(x, y, b)    if \_\_name\_\_ == "\_\_main\_\_":      main() |

Output of above piece of code is:

Estimated coefficients:

b\_0 = -0.0586206896552

b\_1 = 1.45747126437

And graph obtained looks like this:  


## Multiple linear regression

Multiple linear regression attempts to model the relationship between **two or more features** and a response by fitting a linear equation to observed data.

Clearly, it is nothing but an extension of Simple linear regression.

Consider a dataset with **p** features(or independent variables) and one response(or dependent variable).  
Also, the dataset contains **n** rows/observations.

We define:

X (**feature matrix**) = a matrix of size **n X p** where x\_{ij} denotes the values of jth feature for ith observation.

So,

and

y (**response vector**) = a vector of size **n** where y\_{i} denotes the value of response for ith observation.

The **regression line** for **p** features is represented as:  
  
where h(x\_i) is **predicted response value** for ith observation and b\_0, b\_1, …, b\_p are the **regression coefficients**.

Also, we can write:

where e\_i represents **residual error** in ith observation.

We can generalize our linear model a little bit more by representing feature matrix **X** as:  
  
So now, the linear model can be expressed in terms of matrices as:

where,

and

Now, we determine **estimate of b**, i.e. b’ using **Least Squares method**.

As already explained, Least Squares method tends to determine b’ for which total residual error is minimized.

We present the result directly here:  
  
where ‘ represents the transpose of the matrix while -1 represents the matrix inverse.

Knowing the least square estimates, b’, the multiple linear regression model can now be estimated as:

where y’ is **estimated response vector**.

**Note:** The complete derivation for obtaining least square estimates in multiple linear regression can be found here.

Given below is the implementation of multiple linear regression technique on the Boston house pricing datasetdataset using Scikit-learn.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  from sklearn import datasets, linear\_model, metrics    # load the boston dataset  boston = datasets.load\_boston(return\_X\_y=False)    # defining feature matrix(X) and response vector(y)  X = boston.data  y = boston.target    # splitting X and y into training and testing sets  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4,                                                      random\_state=1)    # create linear regression object  reg = linear\_model.LinearRegression()    # train the model using the training sets  reg.fit(X\_train, y\_train)    # regression coefficients  print('Coefficients: \n', reg.coef\_)    # variance score: 1 means perfect prediction  print('Variance score: {}'.format(reg.score(X\_test, y\_test)))    # plot for residual error    ## setting plot style  plt.style.use('fivethirtyeight')    ## plotting residual errors in training data  plt.scatter(reg.predict(X\_train), reg.predict(X\_train) - y\_train,              color = "green", s = 10, label = 'Train data')    ## plotting residual errors in test data  plt.scatter(reg.predict(X\_test), reg.predict(X\_test) - y\_test,              color = "blue", s = 10, label = 'Test data')    ## plotting line for zero residual error  plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)    ## plotting legend  plt.legend(loc = 'upper right')    ## plot title  plt.title("Residual errors")    ## function to show plot  plt.show() |

Output of above program looks like this:

Coefficients:

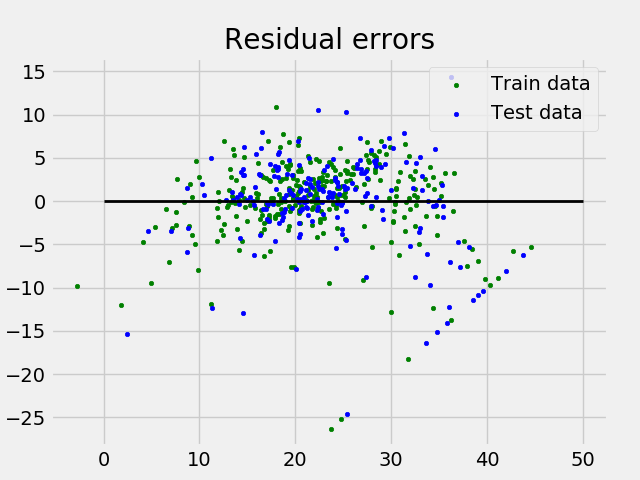
[ -8.80740828e-02 6.72507352e-02 5.10280463e-02 2.18879172e+00

-1.72283734e+01 3.62985243e+00 2.13933641e-03 -1.36531300e+00

2.88788067e-01 -1.22618657e-02 -8.36014969e-01 9.53058061e-03

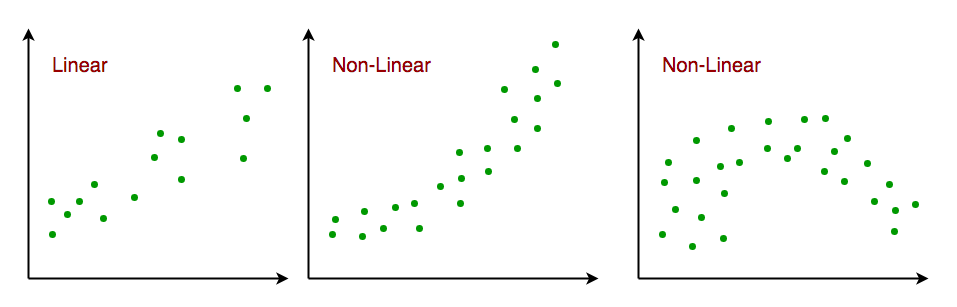
-5.05036163e-01]

Variance score: 0.720898784611

and **Residual Error plot** looks like this:  


## Assumptions

Given below are the basic assumptions that a linear regression model makes regarding a dataset on which it is applied:

* **Linear relationship**: Relationship between response and feature variables should be linear. The linearity assumption can be tested using scatter plots. As shown below, 1st figure represents linearly related variables where as variables in 2nd and 3rd figure are most likely non-linear. So, 1st figure will give better predictions using linear regression.  
  
* **Little or no multi-collinearity**: It is assumed that there is little or no multicollinearity in the data. Multicollinearity occurs when the features (or independent variables) are not independent from each other.
* **Little or no auto-correlation**: Another assumption is that there is little or no autocorrelation in the data. Autocorrelation occurs when the residual errors are not independent from each other. You can refer here for more insight into this topic.
* **Homoscedasticity**: Homoscedasticity describes a situation in which the error term (that is, the “noise” or random disturbance in the relationship between the independent variables and the dependent variable) is the same across all values of the independent variables. As shown below, figure 1 has homoscedasticity while figure 2 has heteroscedasticity.  
  