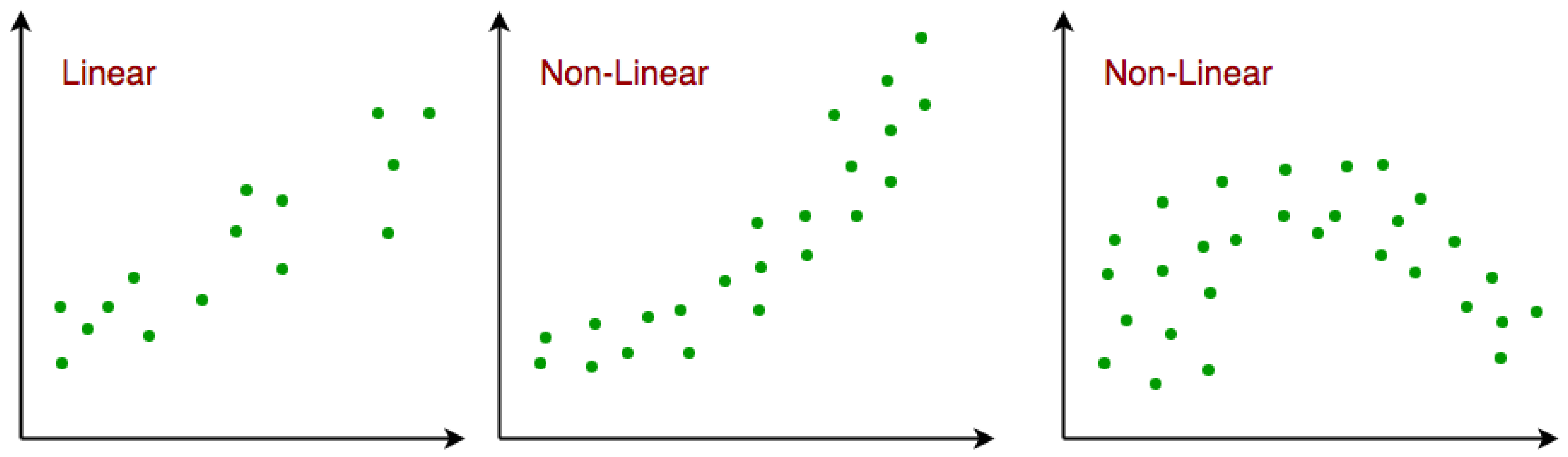
Linear Regression (Python Implementation)

This article discusses the basics of linear regression and its implementation in Python programming language.

Linear Regression is a statistical approach for modeling relationship between a **dependent** variable (**Y**) with a given set of **independent** variables (**X**).

Regression Methods:

- Linear
  - Simple
  - Multiple
- non-Linear

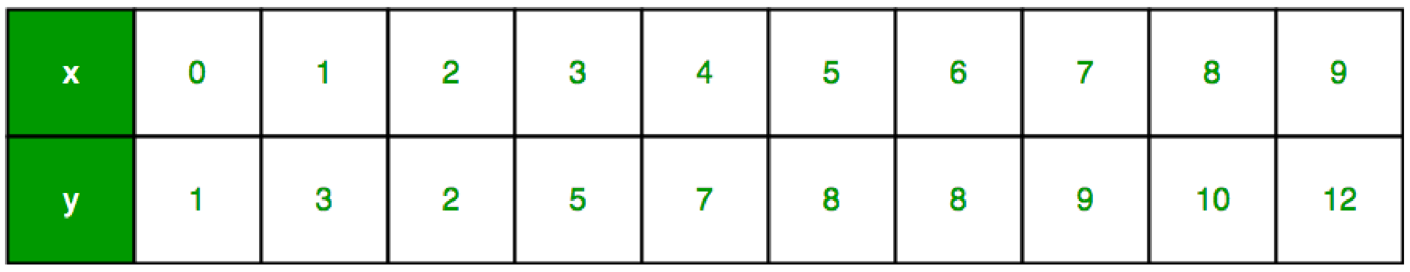


Simple Linear Regression: *Simple Linear Regression* is an approach for predicting a **response** (**Y**) using a single **feature (X)**.

تحلیل گر داده، هیچ گاه مطمین نیست که کدام الگوریتم، مدل (فرمول) دقیق‌تری می دهد.
برای مثال ما، به نظر می‌رسد که داده‌ها با هم ارتباط خطی دارند. به صورت خطی کاهش می یابند. بنابراین، روش رگراسیون خطی ساده را برای پیدا کردن مدل، اجرا می کنیم.

It 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 variable (X)*.

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



x	0	1	2	3	4	5	6	7	8	9
y	1	3	2	5	7	8	8	9	10	12

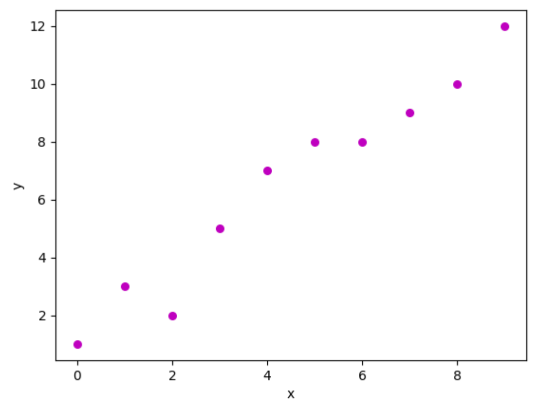
For generality, we define:

X = [x_1, x_2, … , x_n]

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:

$$h(x_i) = \beta_0 + \beta_1 x_i$$

Here,

- $h(x_i)$ represents the **predicted response value** for i th 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:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i = h(x_i) + \varepsilon_i \Rightarrow \varepsilon_i = y_i - h(x_i)$$

Here, ε_i is **residual error** in i th observation. So, our aim is to minimize the total residual error. We define the **squared error** (*cost function*), J as:

$$J(\beta_0 , \beta_1) = \frac{1}{2n} \sum_{i=1}^{n} \varepsilon_i^2$$

and our task is to find the value of β_0 and β_1 for which $J(\beta_0 , \beta_1)$ is minimum!

Without going into the mathematical details[1], we present the result here:

$$SS_{xy} = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^{n} y_i x_i - n\bar{x}\bar{y}$$

where SS_{xy} is the sum of cross-deviations of y and x :

$$SS_{xy} = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^{n} y_i x_i - n\bar{x}\bar{y}$$

and SS_{xx} is the sum of squared deviations of x :

$$SS_{xy} = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^{n} y_i x_i - n\bar{x}\bar{y}$$

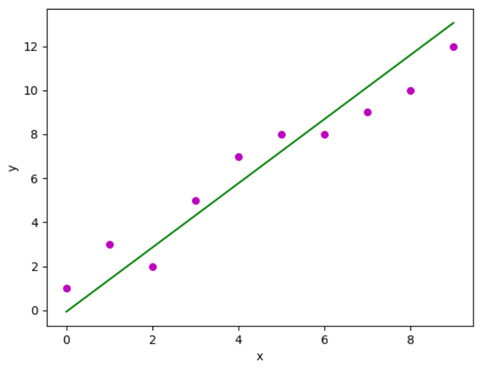
Given below is the python implementation[2] of above technique on our small data-set:

```python
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef (x, y):
# ...
# omitted
# ...
    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:



1 https://www.amherst.edu/system/files/media/1287/SLR_Leastsquares.pdf
2 Complete Code: q3.py