Simple linear regression with Python and R: three ways to begin with

Axel Drefahl | axeleratio@gmail.com | axeleratio.com

Last updated: March 10, 2019

Summary

The basic formulas required to perform a simple linear regression (SLR) analysis are reviewed. A sample data set with calculation results is provided for the purpose of testing and illustration. We demonstrate how an SLR model can be derived in different ways by Python and R programming. The provided code snippets are supposed to serve as hands-on starting points while evaluating relationships between two variables.
This document has been made available at
www.axeleratio.com/math/comp/linreg/linregways.pdf and a brief introduction was posted to the axeleratio blog.

Keywords: Linear regression, Python, NumPy, SciPy, R, free software, programming, statistical computing.

Introduction

Simple linear regression (**SLR**) is a method to test and establish **linear relationships** between two variables. Given a combination of an **independent variable** x and a **dependent variable** y realized by a certain number of sample-value pairs (x_i, y_i), SLR treatment results in an empirical relation of the form

$$y' = a + bx \tag{1}$$

In this equation, a and b are **regression coefficients** and y' is the **response variable**. Assuming a and b being derived with n sample-value pairs ($1 \leq i \leq n$), then y' values are either fitted values y'_i (not necessarily equal to y_i) for the x_i's of the sample set or estimated values y'_{n+k} for new x_{n+k} ($k \geq 1$).

Equation (1) can be considered as a data fitting result, representing the sample data. Depending on the quality and goodness of fit, the equation may turn out as a linear model for the association of the two variables. Such models are useful in predicting dependent quantities or in evaluating the linearity or non-linearity for a two-variable relationship.

Our goal here is to compare ways of deriving such relations or models by programming with the **Python** and **R** languages. Therefore, we summarize the formulas required to calculate regression coefficients in the next section. Also, the calculation of "description-of-fit" values is shown. Further sections provide a data set for testing and explore the use of **NumPy**, **SciPy** and **R** to actually compute SLR values.

Formula background

Introductions to linear regression analysis can be found in many textbooks and online resources. The description and notations, which we use in the following with only slight variations, are based on the work by Edwards [7], Moore [8] and Woodward [12]. In all three texts the **method of least squares** is applied to to explain **data fitting**. The intent of our formula overview herein was made to capture the different terms in use and associate them with a procedure for their calculation—without a detailed discussion elaborating statistical interpretation.

SLR computation expects associated sample values for the independent and dependent variable as input. Let's assume we have a sample set of n data pairs with the values for the variables x and y structured as arrays such that the pair associations is preserved by index i:

$$x = [x_1, \ldots, x_i, \ldots, x_n];\; y = [y_1, \ldots, y_i, \ldots, y_n] \tag{2}$$

Based on these two arrays, we now write down the basic formulas needed to perform SLR analysis. Summation is always over n. For convenience, we drop index subscripts and summation bounds from the formulas.

Means and sums derived therewith. The mean values are:

$$\bar{x} = \frac{\Sigma x}{n},\ \bar{y} = \frac{\Sigma y}{n} \tag{3}$$

We further need the **sums of squares of the values about their means**:

$$S_{xx} = \Sigma(x - \bar{x})^2,\ S_{yy} = \Sigma(y - \bar{y})^2 \tag{4}$$

and the **sum of products of the deviations of the x and y values from the means**:

$$S_{xy} = \Sigma(x - \bar{x})(y - \bar{y}) \tag{5}$$

Regression coefficients. Regression coefficient b, also named **slope** or **gradient**, is

$$b = \frac{S_{xy}}{S_{xx}} = \frac{\Sigma xy - (\Sigma x)(\Sigma y)/n}{\Sigma x^2 - (\Sigma x)^2/n} \tag{6}$$

and regression coefficient a, also known as **intercept**, is

$$a = \bar{y} - b\bar{x} \tag{7}$$

ANOVA sums of squares. The ANalysis Of VAriance (ANOVA) gives a basis for quantifying the variation about the regression and for performing tests of significance. The **total sum of squared deviations of the y values from $\bar{y}$**, $SS_{tot} = S_{yy}$, can be partitioned into the **sum of squares for regression** (also named **sum of squares due to regression**),

$$SS_{reg} = \Sigma(y' - \bar{y})^2 = S_{xy}^2/S_{xx}, \tag{8}$$

and the **residual sum of squares** (also named **sum of squares about regression**),

$$SS_{res} = \Sigma(y - y')^2 = S_{yy} - S_{xy}^2/S_{xx}; \tag{9}$$

such that $SS_{tot} = SS_{reg} + SS_{res}$. In the case of a perfect linear relationship between y and x with all points falling exactly on a straight line ($y = y'$), we will have $SS_{res} = 0$.

ANOVA mean squares. A mean square (MS) equals the sum of squares divided by the associated degree of freedom (DF):

$$MS_{tot} = \frac{SS_{tot}}{n-1},\ MS_{reg} = SS_{reg},\ MS_{res} = \frac{SS_{res}}{n-2} \tag{10}$$

Note that generally $MS_{tot} \neq MS_{reg} + MS_{res}$.

Correlation coefficient. The correlation coefficient between y and x is defined as

$$r = \frac{S_{xy}}{\sqrt{S_{xx}S_{yy}}}. \tag{11}$$

For **r-squared** we have:

$$r^2 = \frac{S_{xy}^2}{S_{xx}S_{yy}} = \frac{SS_{reg}}{SS_{tot}} = 1 - \frac{SS_{res}}{SS_{tot}}. \tag{12}$$

What is called **adjusted r-squared** is calculated as

$$r_{adj}^2 = 1 - \frac{MS_{res}}{MS_{tot}}. \tag{13}$$

Standard errors. The **residual standard error**, also named the **standard error about regression**, is

$$\sigma_{res} = \sqrt{MS_{res}} = \sqrt{\frac{SS_{res}}{n-2}} \tag{14}$$

and σ_{res}^2 is the **variance about regression**. The **standard error of** a is

$$s_a = \sigma_{res}\sqrt{\frac{\Sigma x^2}{nS_{xx}}}. \tag{15}$$

The **standard error of** b is

$$s_b = \frac{\sigma_{res}}{\sqrt{S_{xx}}}. \tag{16}$$

Test-of-significance parameters. The t-test parameter is defined as

$$t = \frac{b}{\sqrt{MS_{res}/S_{xx}}}. \tag{17}$$

The F value is

$$F = \frac{MS_{reg}}{MS_{res}}. \tag{18}$$

Descriptive statistics of residuals (dsr). The residual values, $e_i = y_i - y_i'$, show us how much the fitted y-values deviate from the observed ones. After sorting the residuals such that the values are in ascending order, the **lowest value**, dsrMin, and the largest value, dsrMax, are easily identified. Further descriptors include the **25th percentile** (**1st quartile**), **50th percentile** (**2nd quartile**, better known as **median**) and **75th percentile** (**3rd quartile**). We use the notations dsr25P, dsr50P and dsr75P for these descriptors that give us the residual value markers below which 25%, 50% and 75% of the residual value population, respectively, is found. Note that

$$\text{dsrMin} \leq \text{dsr25P} \leq \text{dsr50P} \leq \text{dsr75P} \leq \text{dsrMax} \tag{19}$$

Calculation Example

In this section we present sample data and calculated SLR values for comparison with output from program runs.

Table 1 contains the sample data. The values are from Woodward's

Table 1: Two physical properties of rubber samples: hardness values (x_i's) and abrasion loss values (y_i's).

i	x_i	y_i	i	x_i	y_i	i	x_i	y_i
1	45	372	11	64	164	21	71	219
2	55	206	12	68	113	22	80	186
3	61	175	13	79	82	23	82	155
4	66	154	14	81	32	24	89	114
5	71	136	15	56	228	25	51	341
6	71	112	16	68	196	26	59	340
7	81	55	17	75	128	27	65	283
8	86	45	18	83	97	28	74	267
9	53	221	19	88	64	29	81	215
10	60	166	20	59	249	30	86	148

Table 2: Calculated mean values, sums, regression and correlation coefficients, errors, and F- and t-statistic values

$\bar{x} = 70.27$, $\bar{y} = 175.4$, $S_{xx} = 4,300$, $S_{yy} = 225,011$, $S_{xy} = -22,946$
$a = 550.4151$, $b = -5.3366$, $s_a = 65.7867$, $s_b = 0.9229$
$r = -0.7377$, $r^2 = 0.5442$, $r_{adj} = 0.7266$, $r^2_{adj} = 0.5279$
$\sigma_{res} = 60.5205$, $F = 33.43$, $t = -5.78$

Table 3: Analysis of Variance (ANOVA)

Source of Variation	Sum of Squares	DF	Mean Square
Due to regression	$SS_{reg} = 122,455$	1	$MS_{reg} = 122,455$
About regression	$SS_{res} = 102,556$	28	$MS_{res} = 3,663$
Total	$SS_{tot} = 225,011$	29	$MS_{tot} = 7,759$

Table 4: Response values $(y - i')$ and residuals $(e_i = y_i - y'_i)$

i	y'_i	e_i	i	y'_i	e_i	i	y'_i	e_i
1	310.27	61.73	11	208.88	-44.88	21	171.52	47.48
2	256.90	-50.90	12	187.53	-74.53	22	123.49	62.51
3	224.89	-49.89	13	128.83	-46.83	23	112.82	42.18
4	198.20	-44.20	14	118.15	**-86.15**	24	75.46	38.54
5	171.52	-35.52	15	251.57	-23.57	25	278.25	62.75
6	171.52	-59.52	16	187.53	8.47	26	235.56	104.44
7	118.15	-63.15	17	150.17	-22.17	27	203.54	79.46
8	91.47	-46.47	18	107.48	-10.48	28	155.51	**111.49**
9	267.58	-46.58	19	80.80	-16.80	29	118.15	96.85
10	230.22	-64.22	20	235.56	13.44	30	91.47	56.53

introduction to “Linear Relationships Between Two Variables” [12]: For thirty rubber specimens the hardness in I.R.H. (Shore) units and the abrasion loss in g./h.p.-hour are given. The table lists values for each specimen (numbered by i), with hardness being the independent variable x and abrasion loss the dependent variable y.

As pointed out by Woodward, measurement of hardness is quickly and simple, while the determination of abrasion loss is more elaborate. Therefore, a method that allows a sufficiently accurate prediction of abrasion loss from hardness would be of considerable advantage.

A CSV file with the data pairs of Table 1 (x_i values separated from y_i values by semicolon) is available at:
www.axeleratio.com/math/comp/linreg/csv/woodward71.csv.

Table 2 lists values calculated based on the formulas given in the previous section. sums and regression and correlation coefficients.

The ANOVA values are given with Table 3. The estimates (response values) and residuals are given in Table 4. The minimum and maximum residuals are in bold type. The values for the descriptive statistics of the residuals are: dsrMin $= -86.15$, dsr25P $= -46.77$, dsr50P $= -19.49$, dsr75P $= 54.27$, dsrMax $= 111.49$.

SLR with NumPy

SLR can be implemented with Python from scratch [3]. Such an approach requires repetitive writing of loops over arrays. NumPy comes with a meta-language for array arithmetic, so we can trim our manual job of algorithmic coding. NumPy is a Python package for scientific computing including a powerful array object [4].

We begin with importing the libraries (packages) we need and then loading the x and y values from a CSV file:

```python
import math, csv
import numpy as np

fcsv  = open(csvFile)
xlst, ylst = [], []
try:
   rows = csv.reader(fcsv, delimiter=';')

   # Get first row with variable (column) names
   headrow = next(rows, None)

   # Continue with rows having semicolon-separated numeric entries
   for row in rows:
      xlst.append(float(row[0]))
      ylst.append(float(row[1]))
finally:
   fcsv.close()

x = np.array(xlst)
y = np.array(ylst)
```

For example, we would assign `csvFile` with the path to file `woodward71.csv`. Note that in the last two lines the lists are placed in NumPy array containers. Now, we are ready to compute the SLR values. This is done by “translating” the formulas given above into Python/NumPy code:

```
n = np.size(x) # should be equal to np.size(y)
xmean, ymean = np.mean(x), np.mean(y)
xd, yd = x - xmean, y - ymean
Sxx, Syy, Sxy = np.sum(xd*xd), np.sum(yd*yd), np.sum(xd*yd)
b = Sxy/Sxx
a = ymean - b * xmean
SSreg = Sxy*Sxy/Sxx
SSres = Syy - SSreg
SStot = Syy # = SSreg + SSres
MStot = SStot/float(n-1)
MSreg = SSreg
MSres = SSres/float(n-2)
r = Sxy/(math.sqrt(Sxx*Syy))
radj2 = 1 - MSres/MStot
sres  = math.sqrt(MSres)
sumx2 = np.sum(x*x)
sa = sres * math.sqrt(sumx2/(float(n)*Sxx))
sb = sres/math.sqrt(Sxx)
t  = b/math.sqrt(MSres/Sxx)
F  = b*b*Sxx/MSres
```

Finally, we calculate the residuals and derive the dsr values by using NumPy's median and percentile methods:

```
residuals = []
i = 0
while i < n:
   yfitted = a + b*xlst[i]
   residuals.append(ylst[i]-yfitted)
   i += 1
dsrMin = min(residuals)
dsr25P = np.percentile(np_res,25)
dsr50P = np.median(res)
dsr75P = np.percentile(np_res,75)
dsrMax = max(residuals)
```

That's it; although, typically, we would insert validation code (for example, before division by `Sxx`) to control program flow and smoothly handle extreme values and unexpected situations. We skipped such code here to focus on the main SLR procedure. With x_i's and y_i's loaded from `woodward71.csv`, a print-out of the assigned variables should display values that match those given in Tables 2 to 4.

SLR with SciPy

The SciPy package complements NumPy [5]. It contains a least-square regression function for two sets of measurements [6]. If you are mainly interested in the regression coefficients and the correlation coefficient, using this function makes it simple to get these values. Before executing the function, the x and y values need to be put into NumPy arrays—as before. In addition, SciPy's `stats` module has to be made available:

```
from scipy import stats
```

Then, it takes only one further line to execute the SLR calculation:

```
b, a, r, p_value, std_err = stats.linregress(x,y)
```

Therewith, we get the gradient (`b`), the slope (`a`) and the correlation coefficient (`r`). Unfortunately, there exists some confusion about what we exactly get with `p_value` and `std_err` [1, 10]. Unless you are sure about their SciPy definition or ready to delve deeper into this issue, we suggest to calculate standard errors as shown above under NumPy only. p-Value and error calculations are also easily obtained via the `lm()` function in R—demonstrated in the next section.

SLR with `lm()` in R

R provides a convenient data structure called **data frame** [11]. It allows the storage of two-dimensional tables of numeric and other data. For example, the data of Table 1 can be framed as `sampledata`:

```
sampledata = data.frame(
  x = c(45, 55, 61, 66, 71, 71, 81, 86, 53, 60,
        64, 68, 79, 81, 56, 68, 75, 83, 88, 59,
        71, 80, 82, 89, 51, 59, 65, 74, 81, 86),
  y = c(372, 206, 175, 154, 136, 112, 55, 45, 221, 166,
        164, 113, 82,  32, 228, 196, 128, 97, 64, 249,
        219, 186, 155, 114, 341, 340, 283, 267, 215, 148)
)
```

Instead of manually typing values, we typically want to import them from a file. Having the values ready in a CSV file (compare with data loading in NumPy), such that each row (line) contains an x_i and a y_i value separated by a semicolon, we use the following command to create data frame `sampledata`:

```
sampledata = read.csv("datafile.csv",header=TRUE, sep=";")
```

With the specification `header=TRUE`, a header row with the variable names is expected: "`x; y`" for x and y. If we use the setting `header=FALSE`, the variable names would be V1 and V2. This is important to remember, since we need to reference correct variable names to continue with the regression.

The function `lm()` fits a linear model to data, which have to be supplied in the data frame or another compatible format [2]. In the following, `lm()` is applied to `sampledata` and the fitted model is saved as an object named `linearmodel`.

```
linearmodel = lm(y ~ x, data = sampledata)
summary(linearmodel)
```

The summary function creates the following output:

```
Call:
lm(formula = y ~ x, data = sampledata)

Residuals:
   Min      1Q Median      3Q     Max
-86.15 -46.77 -19.49   54.27 111.49

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 550.4151    65.7867   8.367 4.22e-09 ***
x            -5.3366     0.9229  -5.782 3.29e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1

Residual standard error: 60.52 on 28 degrees of freedom
Multiple R-squared:  0.5442,    Adjusted R-squared: 0.5279
F-statistic: 33.43 on 1 and 28 DF, p-value: 3.294e-06
```

The output repeats the call of the `lm()` function followed by SLR results explained in the following.
Residuals. `Min`, `1Q`, `Median`, `3Q` and `Max` correspond to dsrMin, dsr25P, dsr50P, dsr75P and dsrMax, respectively.

Coefficients. The "Estimate" column gives the regression coefficients b (Intercept) and a (x row). The standard erros s_b and s_a are listed in the "Std. Error" column. Each value in column "t values" is derived by the division of the "Estimate" value by the "Std. Error" value. These values in combination with the p values in the "Pr($>|t|$)" column are useful in the judgement whether we found a linear relationship or not. The close-to-zero values in the "Pr($>|t|$)"indicate that it is unlikely the found relationship is merely due to chance. The three asterisks at the end of each coefficient row visually support the significance of the relation ship: highly significant p-values and the rejection of the null hypothesis of a simply-by-chance-relationship). The meaning of the asterisks codes are explained in the "Signif. codes" line, which shows the associated statistical significance level.
Residual Standard Error. Residual standard error is σ_{res} of formula 14
Multiple R-squared. Multiple R-squared is r^2 of formula 12.
Adjusted R-squared. Adjusted R-squared is r^2_{adj} of formula 13.
F-statistic. The "F-statistic" value is calculated based on formula 18.

You can individually access the values displayed in the summary. The following example lines show how to get a, b, s_a and s_b via the model summary `ms` by using the function `coef()` (an alias of `coefficients()` [9]):

```
ms = summary(linearmodel)
a  = coef(ms)["(Intercept)","Estimate"]
b  = coef(ms)["x","Estimate"]
sa = coef(ms)["(Intercept","Std. Error"]
sb = coef(ms)["x","Std. Error"]
```

The residual standard error, σ_{res}, and the squared correlation coefficients, r^2 and r^2_{adj}, are extracted from `ms` with these lines:

```
sres  = ms$sigma
r2    = ms$r.squared
radj2 = ms$adj.r.squared
```

Conclusion

Linear regression computation includes multiple operations with array data. The NumPy package provides efficient tools for easily implementing data structures and methods to perform SLR analysis. The `linregress()` function in the `stats` module of the SciPy packages offers a convenient way to directly derive the regression coefficients and the correlation coefficient. Since the assignment of the p-value and standard error values is not as transparent as we would like, we prefer to calculate statistical descriptors by using NumPy functionality only. If you prefer a consistent presentation of the regression model along with statistcal data, you can achieve this in a few lines with R, using `data.frame` and `lm()`. R's versatile `summary` makes it easy to display an SLR summary report as well as extracting result values individually.

If your visual or statistical data analysis suggests a non-linear relationship between the variables of interest, you may want to consider applying curvilinear regression analysis as well.

About the author

Axel Drefahl has designed scientific software for chemical property prediction at the Technical University of Munich, Germany, and Stanford University, California. At the Freiberg University of Mining and Technology he developed Monte-Carlo-simulation algorithms to virtually study interactions of functionalized nanoparticles. Axel initiated the CurlySMILES Project for the encoding of complex, annotated molecular structures, polymer systems and nanoarchitectures. His experience and interests include pattern recognition, nanoinformatics, sustainable chemistry and the history (and future) of science. Off-line, Axel enjoys the outdoors, nature studies and photography. Back online, he shares his findings and impressions on TrailingAhead, Latintos, Explore Reno-Tahoe and other sites.

Literature & Links

[1] Picrin at GitHub. pvalues in scipy.stats.linregress. `https://github.com/scipy/scipy/issues/7074`. Accessed: 2019-02-17.

[2] Ralph at R-bloggers. Simple linear regression. `https://www.r-bloggers.com/simple-linear-regression-2/`. Accessed: 2019-02-17.

[3] Jason Brownlee. How to implement simple linear regression from scratch with Python. `https://machinelearningmastery.com/implement-simple-linear-regression-scratch-python/`. Accessed: 2019-02-17.

[4] The SciPy community. Quickstart tutorial. https://docs.scipy.org/doc/numpy/user/quickstart.html. Accessed: 2019-02-19.

[5] The SciPy community. SciPy tutorial. https://docs.scipy.org/doc/scipy/reference/tutorial/index.html. Accessed: 2019-02-19.

[6] The SciPy community. scipy.stats.linregress. https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.linregress.html. Accessed: 2019-02-19.

[7] A. L. Edwards. The equation of a straight line. In *Multiple Rgression and the Analysis of Variance*, chapter 2, pages 7–30. W. H. Freeman and Company, New York, 2 edition, 1984.

[8] D. S. Moore. Understanding relationships. In *Statistics—Concepts and Controversies*, chapter 5, pages 241–298. W. H. Freeman and Company, New York, 3 edition, 1991.

[9] RDocumentation. coef. https://www.rdocumentation.org/packages/stats/versions/3.5.2/topics/coef. Accessed: 2019-02-19.

[10] Stackoverflow. Definition of standard error in scipy.stats.linregress. https://stackoverflow.com/questions/31455470/definition-of-standard-error-in-scipy-stats-linregress. Accessed: 2019-02-19.

[11] Tutorialspoint. R - data types. https://www.tutorialspoint.com/r/r_data_types.htm. Accessed: 2019-02-17.

[12] R. H. Woodward. Linear relationship between two variables. In O. L. Davies and P. L. Goldsmith, editors, *Statistical Methods in Research and Production*, chapter 7, pages 178–236. Longman, London and New York, 4 edition, 1984.