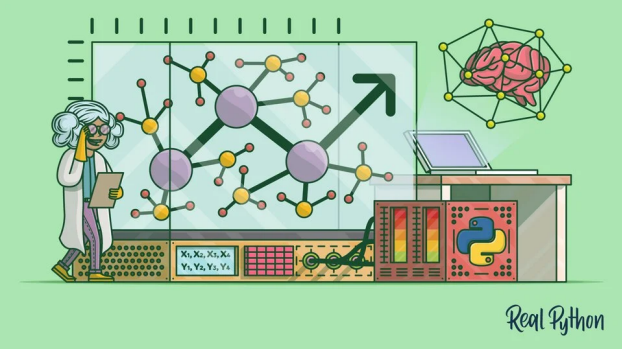
20/12/2021, 13:30 Linear Regression in Python – Real Python



**Linear Regression in Python** by Mirko Stojiljković  20 Comments  **data-science intermediate machine-learning**

Mark as Completed 

Table of Contents

Regression

What Is Regression?

When Do You Need Regression?

Linear Regression

Problem Formulation

Regression Performance

Simple Linear Regression

Multiple Linear Regression

Polynomial Regression

Underfitting and Overfitting

Implementing Linear Regression in Python

Python Packages for Linear Regression

Simple Linear Regression With scikit-learn

Multiple Linear Regression With scikit-learn

Polynomial Regression With scikit-learn

Advanced Linear Regression With statsmodels

Beyond Linear Regression

Conclusion

 **Tweet**  **Share**  **Email**

**** Remove ads

We’re living in the era of large amounts of data, powerful computers, and artificial intelligence. This is just the beginning. Data science and machine learning are driving image recognition, autonomous vehicles development, decisions in the financial and energy sectors, advances in medicine, the rise of social networks, and more. Linear

regression is an important part of this.

**Improve Your Python**

https://realpython.com/linear-regression-in-python/ 1/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

Linear regression is one of the fundamental statistical and machine learning techniques. Whether you want to do statistics, machine learning, or scientific computing, there are good chances that you’ll need it. It’s advisable to learn it first and then proceed towards more complex methods.

**By the end of this article, you’ll have learned:**

What linear regression is

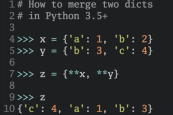
What linear regression is used for

How linear regression works

How to implement linear regression in Python, step by step

**Improve Your Python**

**Free Bonus: Click here to get access to a free NumPy Resources Guide** that points you to the best tutorials,

videos, and books for improving your NumPy skills. **Regression** 

...with a fresh �� **Python Trick** �� code snippet every couple of days:

Email Address

Regression analysis is one of the most important fields in statistics and machine learning. There are many regression

methods available. Linear regression is one of them.

**Send Python Tricks »**

**** Remove ads

What Is Regression?

Regression searches for relationships among variables.

For example, you can observe several employees of some company and try to understand how their salaries depend on the **features**, such as experience, level of education, role, city they work in, and so on.

This is a regression problem where data related to each employee represent one **observation**. The presumption is that the experience, education, role, and city are the independent features, while the salary depends on them.

Similarly, you can try to establish a mathematical dependence of the prices of houses on their areas, numbers of bedrooms, distances to the city center, and so on.

Generally, in regression analysis, you usually consider some phenomenon of interest and have a number of

observations. Each observation has two or more features. Following the assumption that (at least) one of the features depends on the others, you try to establish a relation among them.

In other words, **you need to find a function that maps some features or variables to others sufficiently well**.

The dependent features are called the **dependent variables**, **outputs**, or **responses**.

The independent features are called the **independent variables**, **inputs**, or **predictors**.

Regression problems usually have one continuous and unbounded dependent variable. The inputs, however, can be continuous, discrete, or even categorical data such as gender, nationality, brand, and so on.

It is a common practice to denote the outputs with �� and inputs with ��. If there are two or more independent

variables, they can be represented as the vector �� = (��₁, …, ��ᵣ), where �� is the number of inputs.

When Do You Need Regression?

Typically, you need regression to answer whether and how some phenomenon influences the other or **how several variables are related**. For example, you can use it to determine *if* and *to what extent* the experience or gender impact salaries.

Regression is also useful when you want **to forecast a response** using a new set of predictors. For example, you could

**Improve Your Python**

t t di t l t i it ti f h h ld f th t h i th td t t ti f d

https://realpython.com/linear-regression-in-python/ 2/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

try to predict electricity consumption of a household for the next hour given the outdoor temperature, time of day, and number of residents in that household.

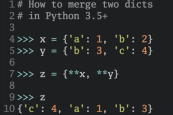
Regression is used in many different fields: economy, computer science, social sciences, and so on. Its importance rises every day with the availability of large amounts of data and increased awareness of the practical value of data.

**Linear Regression**

Linear regression is probably one of the most important and widely used regression techniques. It’s among the simplest regression methods. One of its main advantages is the ease of interpreting results.

Problem Formulation

**Improve Your Python** ...with a fresh �� **Python Trick** ��

When implementing linear regression of some dependent variable �� on the set of independent variables �� = (��₁, …, code snippet every couple of days: 

��ᵣ), where �� is the number of predictors, you assume a linear relationship between �� and ��: �� = ��₀ + ��₁��₁ + ⋯ + ��ᵣ��ᵣ+ ��. This equation is the **regression equation**. ��₀, ��₁, …, ��ᵣ are the **regression coefficients**, and �� is the **random error**. Email Address

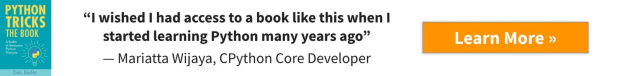
Linear regression calculates the **estimators** of the regression coefficients or simply the **predicted weights**, denoted with ��₀, ��₁, …, ��ᵣ. They define the **estimated regression function** ��(��) = ��₀ + ��₁��₁ + ⋯ + ��ᵣ��ᵣ. This function should

capture the dependencies between the inputs and output sufficiently well.

**Send Python Tricks »**

The **estimated** or **predicted response**, ��(��ᵢ), for each observation �� = 1, …, ��, should be as close as possible to the corresponding **actual response** ��ᵢ. The differences ��ᵢ - ��(��ᵢ) for all observations �� = 1, …, ��, are called the **residuals**. Regression is about determining the **best predicted weights**, that is the weights corresponding to the smallest residuals.

To get the best weights, you usually **minimize the sum of squared residuals** (SSR) for all observations �� = 1, …, ��: SSR = Σᵢ(��ᵢ - ��(��ᵢ))². This approach is called the **method of ordinary least squares**.

 Remove ads

Regression Performance

The variation of actual responses ��ᵢ, �� = 1, …, ��, occurs partly due to the dependence on the predictors ��ᵢ. However, there is also an additional inherent variance of the output.

The **coefficient of determination**, denoted as ��², tells you which amount of variation in �� can be explained by the dependence on �� using the particular regression model. Larger ��² indicates a better fit and means that the model can better explain the variation of the output with different inputs.

The value ��² = 1 corresponds to SSR = 0, that is to the **perfect fit** since the values of predicted and actual responses fit completely to each other.

Simple Linear Regression

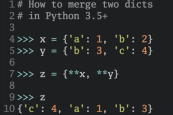
Simple or single-variate linear regression is the simplest case of linear regression with a single independent variable, �� = ��.

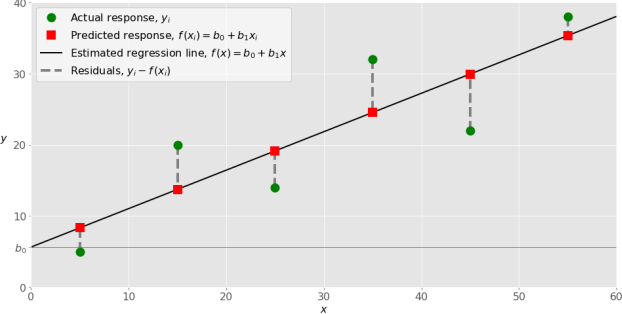
The following figure illustrates simple linear regression:

**Improve Your Python**

https://realpython.com/linear-regression-in-python/ 3/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

Example of simple linear regression

**Improve Your Python** 

...with a fresh �� **Python Trick** �� code snippet every couple of days:

Email Address

**Send Python Tricks »**

When implementing simple linear regression, you typically start with a given set of input-output (��-��) pairs (green circles). These pairs are your observations. For example, the leftmost observation (green circle) has the input �� = 5 and the actual output (response) �� = 5. The next one has �� = 15 and �� = 20, and so on.

The estimated regression function (black line) has the equation ��(��) = ��₀ + ��₁��. Your goal is to calculate the optimal values of the predicted weights ��₀ and ��₁ that minimize SSR and determine the estimated regression function. The value of ��₀, also called the **intercept**, shows the point where the estimated regression line crosses the �� axis. It is the value of the estimated response ��(��) for �� = 0. The value of ��₁ determines the **slope** of the estimated regression line.

The predicted responses (red squares) are the points on the regression line that correspond to the input values. For example, for the input �� = 5, the predicted response is ��(5) = 8.33 (represented with the leftmost red square).

The residuals (vertical dashed gray lines) can be calculated as ��ᵢ - ��(��ᵢ) = ��ᵢ - ��₀ - ��₁��ᵢ for �� = 1, …, ��. They are the distances between the green circles and red squares. When you implement linear regression, you are actually trying to minimize these distances and make the red squares as close to the predefined green circles as possible.

Multiple Linear Regression

Multiple or multivariate linear regression is a case of linear regression with two or more independent variables.

If there are just two independent variables, the estimated regression function is ��(��₁, ��₂) = ��₀ + ��₁��₁ + ��₂��₂. It represents a regression plane in a three-dimensional space. The goal of regression is to determine the values of the weights ��₀, ��₁, and ��₂ such that this plane is as close as possible to the actual responses and yield the minimal SSR.

The case of more than two independent variables is similar, but more general. The estimated regression function is ��(��₁, …, ��ᵣ) = ��₀ + ��₁��₁ + ⋯ +��ᵣ��ᵣ, and there are �� + 1 weights to be determined when the number of inputs is ��.

Polynomial Regression

You can regard polynomial regression as a generalized case of linear regression. You assume the polynomial dependence between the output and inputs and, consequently, the polynomial estimated regression function.

In other words, in addition to linear terms like ��₁��₁, your regression function �� can include non-linear terms such as ��₂��₁², ��₃��₁³, or even ��₄��₁��₂, ��₅��₁²��₂, and so on.

The simplest example of polynomial regression has a single independent variable, and the estimated regression function is a polynomial of degree 2: ��(��) = ��₀ + ��₁�� + ��₂��².

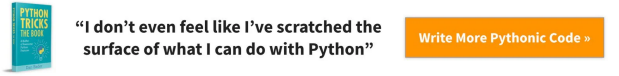
Now, remember that you want to calculate ��₀, ��₁, and ��₂, which minimize SSR. These are your unknowns!

Keeping this in mind, compare the previous regression function with the function ��(��₁, ��₂) = ��₀ + ��₁��₁ + ��₂��₂ used for linear regression. They look very similar and are both linear functions of the unknowns ��₀, ��₁, and ��₂. This is why you can **solve the polynomial regression problem as a linear probImlepmro**w**ve**ith**You**th**r**e**Py**te**th**rm**on**��² regarded as an input variable.

https://realpython.com/linear-regression-in-python/ 4/22

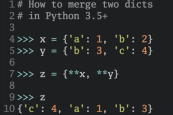
20/12/2021, 13:30 Linear Regression in Python – Real Python

In the case of two variables and the polynomial of degree 2, the regression function has this form: ��(��₁, ��₂) = ��₀ + ��₁��₁ + ��₂��₂ + ��₃��₁² + ��₄��₁��₂ + ��₅��₂². The procedure for solving the problem is identical to the previous case. You apply linear regression for five inputs: ��₁, ��₂, ��₁², ��₁��₂, and ��₂². What you get as the result of regression are the values of six weights which minimize SSR: ��₀, ��₁, ��₂, ��₃, ��₄, and ��₅.

Of course, there are more general problems, but this should be enough to illustrate the point. 

 Remove ads

Underfitting and Overfitting

**Improve Your Python** 

...with a fresh �� **Python Trick** �� code snippet every couple of days:

One very important question that might arise when you’re implementing polynomial regression is related to **the**

**choice of the optimal degree of the polynomial regression function**.

Email Address

There is no straightforward rule for doing this. It depends on the case. You should, however, be aware of two

**Send Python Tricks »**

problems that might follow the choice of the degree: **underfitting** and **overfitting**.

**Underfitting** occurs when a model can’t accurately capture the dependencies among data, usually as a consequence of its own simplicity. It often yields a low ��² with known data and bad generalization capabilities when applied with new data.

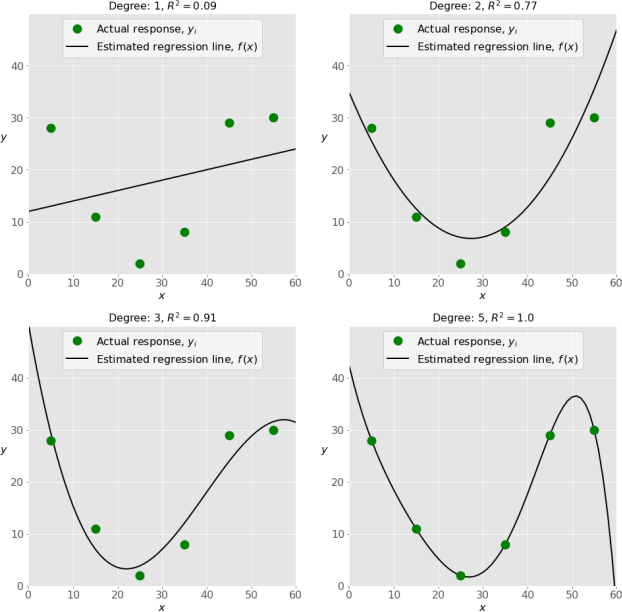
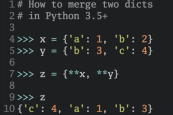
**Overfitting** happens when a model learns both dependencies among data and random fluctuations. In other words, a model learns the existing data too well. Complex models, which have many features or terms, are often prone to overfitting. When applied to known data, such models usually yield high ��². However, they often don’t generalize well and have significantly lower ��² when used with new data.

The next figure illustrates the underfitted, well-fitted, and overfitted models:

**Improve Your Python**

https://realpython.com/linear-regression-in-python/ 5/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

**Improve Your Python** 

...with a fresh �� **Python Trick** ��

code snippet every couple of days:

Email Address

**Send Python Tricks »**

Example of underfitted, well-fitted and overfitted models

The top left plot shows a linear regression line that has a low ��². It might also be important that a straight line can’t take into account the fact that the actual response increases as �� moves away from 25 towards zero. This is likely an example of underfitting.

The top right plot illustrates polynomial regression with the degree equal to 2. In this instance, this might be the

optimal degree for modeling this data. The model has a value of ��² that is satisfactory in many cases and shows

trends nicely.

The bottom left plot presents polynomial regression with the degree equal to 3. The value of ��² is higher than in the preceding cases. This model behaves better with known data than the previous ones. However, it shows some signs of overfitting, especially for the input values close to 60 where the line starts decreasing, although actual data don’t show that.

Finally, on the bottom right plot, you can see the perfect fit: six points and the polynomial line of the degree 5 (or higher) yield ��² = 1. Each actual response equals its corresponding prediction.

In some situations, this might be exactly what you’re looking for. In many cases, however, this is an overfitted model. It is likely to have poor behavior with unseen data, especially with the inputs larger than 50.

For example, it assumes, without any evidence, that there is a significant drop in responses for �� > 50 and that ��

reaches zero for �� near 60. Such behavior is the consequence of excessive effort to learn and fit the existing data.

There are a lot of resources where you can find more information about regression in general and linear regression in particular. The regression analysis page on Wikipedia, Wikipedia’s linear regression article, as well as Khan Academy’s linear regression article are good starting points.

**Improve Your Python**

**Implementing Linear Regression in Python**

https://realpython.com/linear-regression-in-python/ 6/22

**Implementing Linear Regression in Python**

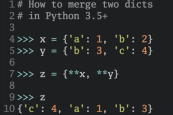
20/12/2021, 13:30 Linear Regression in Python – Real Python

It’s time to start implementing linear regression in Python. Basically, all you should do is apply the proper packages and their functions and classes.

Python Packages for Linear Regression

The package **NumPy** is a fundamental Python scientific package that allows many high-performance operations on single- and multi-dimensional arrays. It also offers many mathematical routines. Of course, it’s open source.

If you’re not familiar with NumPy, you can use the official NumPy User Guide and read Look Ma, No For-Loops: Array

Programming With NumPy. In addition, Pure Python vs NumPy vs TensorFlow Performance Comparison can give you **Improve Your Python** 

a pretty good idea on the performance gains you can achieve when applying NumPy.

The package **scikit-learn** is a widely used Python library for machine learning, built on top of NumPy and some other

...with a fresh �� **Python Trick** ��

code snippet every couple of days:

packages. It provides the means for preprocessing data, reducing dimensionality, implementing regression, classification, clustering, and more. Like NumPy, scikit-learn is also open source.

Email Address

You can check the page Generalized Linear Models on the scikit-learn web site to learn more about linear models and

get deeper insight into how this package works.

**Send Python Tricks »**

If you want to implement linear regression and need the functionality beyond the scope of scikit-learn, you should consider **statsmodels**. It’s a powerful Python package for the estimation of statistical models, performing tests, and more. It’s open source as well.

You can find more information on statsmodels on its official web site.

 Remove ads

Simple Linear Regression With scikit-learn

Let’s start with the simplest case, which is simple linear regression.

There are five basic steps when you’re implementing linear regression:

1. Import the packages and classes you need.

2. Provide data to work with and eventually do appropriate transformations.

3. Create a regression model and fit it with existing data.

4. Check the results of model fitting to know whether the model is satisfactory.

5. Apply the model for predictions.

These steps are more or less general for most of the regression approaches and implementations.

**Step 1: Import packages and classes**

The first step is to import the package numpy and the class LinearRegression from sklearn.linear\_model: Python

import numpy as np

from sklearn.linear\_model import LinearRegression

Now, you have all the functionalities you need to implement linear regression.

The fundamental data type of NumPy is the array type called numpy.ndarray. The rest of this article uses the term **array** to refer to instances of the type numpy.ndarray.

The class sklearn.linear\_model.LinearRegression will be used to perform linear and polynomial regression and make predictions accordingly.

**Step 2: Provide data**

**Improve Your Python**

https://realpython.com/linear-regression-in-python/ 7/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

The second step is defining data to work with. The inputs (regressors, ��) and output (predictor, ��) should be arrays (the instances of the class numpy.ndarray) or similar objects. This is the simplest way of providing data for regression:

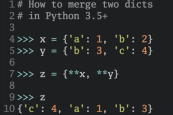
Python

x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))

y = np.array([5, 20, 14, 32, 22, 38])

Now, you have two arrays: the input x and output y. You should call .reshape() on x because this array is required to be **two-dimensional**, or to be more precise, to have **one column and as many rows as necessary**. That’s exactly what the argument (-1, 1) of .reshape() specifies.

This is how x and y look now:

**Improve Your Python** ...with a fresh �� **Python Trick** �� 

Python >>> code snippet every couple of days:

>>> print(x)

[[ 5]

[15]

[25]

[35]

[45]

[55]]

>>> print(y)

[ 5 20 14 32 22 38]

Email Address

**Send Python Tricks »**

As you can see, x has two dimensions, and x.shape is (6, 1), while y has a single dimension, and y.shape is (6,). **Step 3: Create a model and fit it**

The next step is to create a linear regression model and fit it using the existing data.

Let’s create an instance of the class LinearRegression, which will represent the regression model: Python

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 ��₀ (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.

This example uses the default values of all parameters.

It’s time to start using the model. First, you need to call .fit() on model:

Python

model.fit(x, y)

With .fit(), you calculate the optimal values of the weights ��₀ and ��₁, 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. That’s why you can replace the last two statements with this one:

Python

model = LinearRegression().fit(x, y)

This statement does the same thing as the previous two. It’s just shorter.

**Improve Your Python**

https://realpython.com/linear-regression-in-python/ 8/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

**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 (��²) with .score() called on model:

Python >>>

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

>>> print('coefficient of determination:', r\_sq)

coefficient of determination: 0.715875613747954

**Improve Your Python** 

When you’re applying .score(), the arguments are also the predictor x and regressor y, and the return value is ��². ...with a fresh �� **Python Trick** ��

The attributes of model are .intercept\_, which represents the coefficient, ��₀ and .coef\_, which represents ��₁: code snippet every couple of days:

Python >>> Email Address

>>> print('intercept:', model.intercept\_)

intercept: 5.633333333333329

>>> print('slope:', model.coef\_) slope: [0.54]

**Send Python Tricks »**

The code above illustrates how to get ��₀ and ��₁. You can notice that .intercept\_ is a scalar, while .coef\_ is an array.

The value ��₀ = 5.63 (approximately) illustrates that your model predicts the response 5.63 when �� is zero. The value ��₁ = 0.54 means that the predicted response rises by 0.54 when �� is increased by one.

You should notice that you can provide y as a two-dimensional array as well. In this case, you’ll get a similar result. This is how it might look:

Python >>>

>>> new\_model = LinearRegression().fit(x, y.reshape((-1, 1)))

>>> print('intercept:', new\_model.intercept\_)

intercept: [5.63333333]

>>> print('slope:', new\_model.coef\_)

slope: [[0.54]]

As you can see, this example is very similar to the previous one, but in this case, .intercept\_ is a one-dimensional array with the single element ��₀, and .coef\_ is a two-dimensional array with the single element ��₁.

**Step 5: Predict response**

Once there is a satisfactory model, you can use it for predictions with either existing or new data. To obtain the predicted response, use .predict():

Python >>>

>>> y\_pred = model.predict(x)

>>> print('predicted response:', y\_pred, sep='\n')

predicted response:

[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]

When applying .predict(), you pass the regressor as the argument and get the corresponding predicted response. This is a nearly identical way to predict the response:

Python >>>

>>> y\_pred = model.intercept\_ + model.coef\_ \* x

>>> print('predicted response:', y\_pred, sep='\n')

predicted response:

[[ 8.33333333]

[13.73333333]

[19.13333333]**Improve Your Python**

https://realpython.com/linear-regression-in-python/ 9/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

[24.53333333]

[29.93333333]

[35.33333333]]

In this case, you multiply each element of x with model.coef\_ and add model.intercept\_ to the product.

The output here differs from the previous example only in dimensions. The predicted response is now a two dimensional array, while in the previous case, it had one dimension.

If you reduce the number of dimensions of x to one, these two approaches will yield the same result. You can do this by replacing x with x.reshape(-1), x.flatten(), or x.ravel() when multiplying it with model.coef\_.

**Improve Your Python**

In practice, regression models are often applied for forecasts. This means that you can use fitted models to calculate

the outputs based on some other, new inputs:

...with a fresh �� **Python Trick** �� code snippet every couple of days:

Python >>>

>>> x\_new = np.arange(5).reshape((-1, 1))

>>> print(x\_new)

[[0]

[1]

[2]

[3]

[4]]

>>> y\_new = model.predict(x\_new)

>>> print(y\_new)

[5.63333333 6.17333333 6.71333333 7.25333333 7.79333333]

Email Address

**Send Python Tricks »**

Here .predict() is applied to the new regressor x\_new and yields the response y\_new. This example conveniently uses arange() from numpy to generate an array with the elements from 0 (inclusive) to 5 (exclusive), that is 0, 1, 2, 3, and 4.

You can find more information about LinearRegression on the official documentation page.  Remove ads

Multiple Linear Regression With scikit-learn

You can implement multiple linear regression following the same steps as you would for simple regression. **Steps 1 and 2: Import packages and classes, and provide data**

First, you import numpy and sklearn.linear\_model.LinearRegression and provide known inputs and output: Python

import numpy as np

from sklearn.linear\_model import LinearRegression

x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]] y = [4, 5, 20, 14, 32, 22, 38, 43]

x, y = np.array(x), np.array(y)

That’s a simple way to define the input x and output y. You can print x and y to see how they look now: Python >>>

>>> print(x)

[[ 0 1]

[ 5 1]

[15 2]

[25 5]

[35 11]

[45 15]

[55 34] [60 35]]

**Improve Your Python**

https://realpython.com/linear-regression-in-python/ 10/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

[60 35]]

>>> print(y)

[ 4 5 20 14 32 22 38 43]

In multiple linear regression, x is a two-dimensional array with at least two columns, while y is usually a one dimensional array. This is a simple example of multiple linear regression, and x has exactly two columns.

**Step 3: Create a model and fit it**

The next step is to create the regression model as an instance of LinearRegression and fit it with .fit(): Python

model = LinearRegression().fit(x, y)

**Improve Your Python** ...with a fresh �� **Python Trick** ��

The result of this statement is the variable model referring to the object of type LinearRegression. It represents the

regression model fitted with existing data. **Step 4: Get results**

code snippet every couple of days: Email Address 

You can obtain the properties of the model the same way as in the case of simple linear regression: **Send Python Tricks »**

Python >>>

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

>>> print('coefficient of determination:', r\_sq)

coefficient of determination: 0.8615939258756776

>>> print('intercept:', model.intercept\_)

intercept: 5.52257927519819

>>> print('slope:', model.coef\_)

slope: [0.44706965 0.25502548]

You obtain the value of ��² using .score() and the values of the estimators of regression coefficients with .intercept\_ and .coef\_. Again, .intercept\_ holds the bias ��₀, while now .coef\_ is an array containing ��₁ and ��₂ respectively.

In this example, the intercept is approximately 5.52, and this is the value of the predicted response when ��₁ = ��₂ = 0. The increase of ��₁ by 1 yields the rise of the predicted response by 0.45. Similarly, when ��₂ grows by 1, the response rises by 0.26.

**Step 5: Predict response**

Predictions also work the same way as in the case of simple linear regression:

Python >>>

>>> y\_pred = model.predict(x)

>>> print('predicted response:', y\_pred, sep='\n')

predicted response:

[ 5.77760476 8.012953 12.73867497 17.9744479 23.97529728 29.4660957

38.78227633 41.27265006]

The predicted response is obtained with .predict(), which is very similar to the following: Python >>>

>>> y\_pred = model.intercept\_ + np.sum(model.coef\_ \* x, axis=1)

>>> print('predicted response:', y\_pred, sep='\n')

predicted response:

[ 5.77760476 8.012953 12.73867497 17.9744479 23.97529728 29.4660957

38.78227633 41.27265006]

You can predict the output values by multiplying each column of the input with the appropriate weight, summing the results and adding the intercept to the sum.

You can apply this model to new data as well:

Python >>>

>>> x\_new = np.arange(10).reshape((-1, 2))

>>> print(x\_new)**Improve Your Python**

https://realpython.com/linear-regression-in-python/ 11/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

[[0 1]

[2 3]

[4 5]

[6 7]

[8 9]]

>>> y\_new = model.predict(x\_new)

>>> print(y\_new)

[ 5.77760476 7.18179502 8.58598528 9.99017554 11.3943658 ]

That’s the prediction using a linear regression model.

 Remove ads 

Polynomial Regression With scikit-learn

**Improve Your Python**

...with a fresh �� **Python Trick** �� code snippet every couple of days:

Email Address

Implementing polynomial regression with scikit-learn is very similar to linear regression. There is only one extra step:

you need to transform the array of inputs to include non-linear terms such as ��². **Step 1: Import packages and classes**

**Send Python Tricks »**

In addition to numpy and sklearn.linear\_model.LinearRegression, you should also import the class PolynomialFeatures from sklearn.preprocessing:

Python

import numpy as np

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

The import is now done, and you have everything you need to work with.

**Step 2a: Provide data**

This step defines the input and output and is the same as in the case of linear regression: Python

x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))

y = np.array([15, 11, 2, 8, 25, 32])

Now you have the input and output in a suitable format. Keep in mind that you need the input to be a **two dimensional array**. That’s why .reshape() is used.

**Step 2b: Transform input data**

This is the **new step** you need to implement for polynomial regression!

As you’ve seen earlier, you need to include ��² (and perhaps other terms) as additional features when implementing polynomial regression. For that reason, you should transform the input array x to contain the additional column(s) with the values of ��² (and eventually more features).

It’s possible to transform the input array in several ways (like using insert() from numpy), but the class PolynomialFeatures is very convenient for this purpose. Let’s create an instance of this class:

Python

transformer = PolynomialFeatures(degree=2, include\_bias=False)

The variable transformer refers to an instance of PolynomialFeatures which you can use to transform the input x. You can provide several optional parameters to PolynomialFeatures:

**degree** is an integer (2 by default) that represents the degree of the polynomial regression function. **Improve Your Python**

**interaction\_only** is a Boolean (False by default) that decides whether to include only interaction features

https://realpython.com/linear-regression-in-python/ 12/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

(True) or all features (False).

**include\_bias** is a Boolean (True by default) that decides whether to include the bias (intercept) column of ones (True) or not (False).

This example uses the default values of all parameters, but you’ll sometimes want to experiment with the degree of the function, and it can be beneficial to provide this argument anyway.

Before applying transformer, you need to fit it with .fit():

Python

transformer.fit(x) 

**Improve Your Python**

Once transformer is fitted, it’s ready to create a new, modified input. You apply .transform() to do that: ...with a fresh �� **Python Trick** ��

Python

x\_ = transformer.transform(x)

code snippet every couple of days: Email Address

That’s the transformation of the input array with .transform(). It takes the input array as the argument and returns

the modified array.

**Send Python Tricks »**

You can also use .fit\_transform() to replace the three previous statements with only one: Python

x\_ = PolynomialFeatures(degree=2, include\_bias=False).fit\_transform(x)

That’s fitting and transforming the input array in one statement with .fit\_transform(). It also takes the input array and effectively does the same thing as .fit() and .transform() called in that order. It also returns the modified array. This is how the new input array looks:

Python >>>

>>> print(x\_)

[[ 5. 25.]

[ 15. 225.]

[ 25. 625.]

[ 35. 1225.]

[ 45. 2025.]

[ 55. 3025.]]

The modified input array contains two columns: one with the original inputs and the other with their squares. You can find more information about PolynomialFeatures on the official documentation page. **Step 3: Create a model and fit it**

This step is also the same as in the case of linear regression. You create and fit the model: Python

model = LinearRegression().fit(x\_, y)

The regression model is now created and fitted. It’s ready for application.

You should keep in mind that the first argument of .fit() is the *modified input array* x\_ and not the original x. **Step 4: Get results**

You can obtain the properties of the model the same way as in the case of linear regression: Python >>>

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

>>> print('coefficient of determination:', r\_sq)

coefficient of determination: 0.8908516262498564 i t('i t t ' d l i t t )

**Improve Your Python**

https://realpython.com/linear-regression-in-python/ 13/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

>>> print('intercept:', model.intercept\_)

intercept: 21.372321428571425

>>> print('coefficients:', model.coef\_)

coefficients: [-1.32357143 0.02839286]

Again, .score() returns ��². Its first argument is also the modified input x\_, not x. The values of the weights are associated to .intercept\_ and .coef\_: .intercept\_ represents ��₀, while .coef\_ references the array that contains ��₁ and ��₂respectively.

You can obtain a very similar result with different transformation and regression arguments:

Python 

**Improve Your Python**

x\_ = PolynomialFeatures(degree=2, include\_bias=True).fit\_transform(x)

...with a fresh �� **Python Trick** ��

If you call PolynomialFeatures with the default parameter include\_bias=True (or if you just omit it), you’ll obtain

code snippet every couple of days:

the new input array x\_ with the additional leftmost column containing only ones. This column corresponds to the

intercept. This is how the modified input array looks in this case:

Email Address

Python >>>

>>> print(x\_)

[[1.000e+00 5.000e+00 2.500e+01] [1.000e+00 1.500e+01 2.250e+02] [1.000e+00 2.500e+01 6.250e+02] [1.000e+00 3.500e+01 1.225e+03] [1.000e+00 4.500e+01 2.025e+03] [1.000e+00 5.500e+01 3.025e+03]]

**Send Python Tricks »**

The first column of x\_ contains ones, the second has the values of x, while the third holds the squares of x.

The intercept is already included with the leftmost column of ones, and you don’t need to include it again when creating the instance of LinearRegression. Thus, you can provide fit\_intercept=False. This is how the next statement looks:

Python

model = LinearRegression(fit\_intercept=False).fit(x\_, y)

The variable model again corresponds to the new input array x\_. Therefore x\_ should be passed as the first argument instead of x.

This approach yields the following results, which are similar to the previous case:

Python >>>

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

>>> print('coefficient of determination:', r\_sq)

coefficient of determination: 0.8908516262498565

>>> print('intercept:', model.intercept\_)

intercept: 0.0

>>> print('coefficients:', model.coef\_)

coefficients: [21.37232143 -1.32357143 0.02839286]

You see that now .intercept\_ is zero, but .coef\_ actually contains ��₀ as its first element. Everything else is the same. **Step 5: Predict response**

If you want to get the predicted response, just use .predict(), but remember that the argument should be the modified input x\_ instead of the old x:

Python >>>

>>> y\_pred = model.predict(x\_)

>>> print('predicted response:', y\_pred, sep='\n')

predicted response:

[15.46428571 7.90714286 6.02857143 9.82857143 19.30714286 34.46428571]

**Improve Your Python**

As you can see, the prediction works almost the same way as in the case of linear regression. It just requires the

https://realpython.com/linear-regression-in-python/ 14/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

modified input instead of the original.

You can apply the identical procedure if you have **several input variables**. You’ll have an input array with more than one column, but everything else is the same. Here is an example:

**Improve Your Python** 

...with a fresh �� **Python Trick** ��

code snippet every couple of days:

Email Address

**Send Python Tricks »**

Python

# Step 1: Import packages

import numpy as np

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

# Step 2a: Provide data

x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]

y = [4, 5, 20, 14, 32, 22, 38, 43]

x, y = np.array(x), np.array(y)

# Step 2b: Transform input data

x\_ = PolynomialFeatures(degree=2, include\_bias=False).fit\_transform(x)

# Step 3: Create a model and fit it

model = LinearRegression().fit(x\_, y)

# Step 4: Get results

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

intercept, coefficients = model.intercept\_, model.coef\_

# Step 5: Predict

y\_pred = model.predict(x\_)

This regression example yields the following results and predictions:

Python >>>

>>> print('coefficient of determination:', r\_sq)

coefficient of determination: 0.9453701449127822

>>> print('intercept:', intercept)

intercept: 0.8430556452395734

>>> print('coefficients:', coefficients, sep='\n')

coefficients:

[ 2.44828275 0.16160353 -0.15259677 0.47928683 -0.4641851 ]

>>> print('predicted response:', y\_pred, sep='\n')

predicted response:

[ 0.54047408 11.36340283 16.07809622 15.79139 29.73858619 23.50834636

39.05631386 41.92339046]

In this case, there are six regression coefficients (including the intercept), as shown in the estimated regression

function ��(��₁, ��₂) = ��₀ + ��₁��₁ + ��₂��₂ + ��₃��₁² + ��₄��₁��₂ + ��₅��₂².

**Improve Your Python**

Y l ti th t l i l i i ld d hi h ffi i t f d t i ti th lti l li

https://realpython.com/linear-regression-in-python/ 15/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

You can also notice that polynomial regression yielded a higher coefficient of determination than multiple linear regression for the same problem. At first, you could think that obtaining such a large ��² is an excellent result. It might be.

However, in real-world situations, having a complex model and ��² very close to 1 might also be a sign of overfitting. To check the performance of a model, you should test it with new data, that is with observations not used to fit (train) the model. To learn how to split your dataset into the training and test subsets, check out Split Your Dataset With scikit-learn’s train\_test\_split().

**Improve Your Python**

 Remove ads 

Advanced Linear Regression With statsmodels

...with a fresh �� **Python Trick** �� code snippet every couple of days:

Email Address

You can implement linear regression in Python relatively easily by using the package statsmodels as well. Typically,

this is desirable when there is a need for more detailed results. The procedure is similar to that of scikit-learn.

**Step 1: Import packages**

**Send Python Tricks »**

First you need to do some imports. In addition to numpy, you need to import statsmodels.api: Python

import numpy as np

import statsmodels.api as sm

Now you have the packages you need.

**Step 2: Provide data and transform inputs**

You can provide the inputs and outputs the same way as you did when you were using scikit-learn: Python

x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]] y = [4, 5, 20, 14, 32, 22, 38, 43]

x, y = np.array(x), np.array(y)

The input and output arrays are created, but the job is not done yet.

You need to add the column of ones to the inputs if you want statsmodels to calculate the intercept ��₀. It doesn’t takes ��₀ into account by default. This is just one function call:

Python

x = sm.add\_constant(x)

That’s how you add the column of ones to x with add\_constant(). It takes the input array x as an argument and returns a new array with the column of ones inserted at the beginning. This is how x and y look now:

Python >>>

>>> print(x)

[[ 1. 0. 1.]

[ 1. 5. 1.]

[ 1. 15. 2.]

[ 1. 25. 5.]

[ 1. 35. 11.]

[ 1. 45. 15.]

[ 1. 55. 34.]

[ 1. 60. 35.]]

>>> print(y)

[ 4 5 20 14 32 22 38 43]

**Improve Your Python**

https://realpython.com/linear-regression-in-python/ 16/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

You can see that the modified x has three columns: the first column of ones (corresponding to ��₀ and replacing the intercept) as well as two columns of the original features.

**Step 3: Create a model and fit it**

The regression model based on ordinary least squares is an instance of the class

statsmodels.regression.linear\_model.OLS. This is how you can obtain one:

Python

model = sm.OLS(y, x)

**Improve Your Python**

You should be careful here! Please, notice that the first argument is the output, followed with the input. There are

several more optional parameters.

...with a fresh �� **Python Trick** �� code snippet every couple of days: 

To find more information about this class, please visit the official documentation page.

Once your model is created, you can apply .fit() on it:

Python

results = model.fit()

Email Address

**Send Python Tricks »**

By calling .fit(), you obtain the variable results, which is an instance of the class

statsmodels.regression.linear\_model.RegressionResultsWrapper. This object holds a lot of information about the regression model.

**Step 4: Get results**

The variable results refers to the object that contains detailed information about the results of linear regression. Explaining them is far beyond the scope of this article, but you’ll learn here how to extract them.

You can call .summary() to get the table with the results of linear regression:

Python >>>

>>> print(results.summary())

OLS Regression Results

==============================================================================

Dep. Variable: y R-squared: 0.862

Model: OLS Adj. R-squared: 0.806

Method: Least Squares F-statistic: 15.56

Date: Sun, 17 Feb 2019 Prob (F-statistic): 0.00713

Time: 19:15:07 Log-Likelihood: -24.316

No. Observations: 8 AIC: 54.63

Df Residuals: 5 BIC: 54.87

Df Model: 2

Covariance Type: nonrobust

==============================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------

const 5.5226 4.431 1.246 0.268 -5.867 16.912

x1 0.4471 0.285 1.567 0.178 -0.286 1.180

x2 0.2550 0.453 0.563 0.598 -0.910 1.420

==============================================================================

Omnibus: 0.561 Durbin-Watson: 3.268

Prob(Omnibus): 0.755 Jarque-Bera (JB): 0.534

Skew: 0.380 Prob(JB): 0.766

Kurtosis: 1.987 Cond. No. 80.1

==============================================================================

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

This table is very comprehensive. You can find many statistical values associated with linear regression including ��², ��₀, ��₁, and ��₂.

In this particular case, you might obtain the warning related to kurtosistest. This is due to the small number of observations provided.

**Improve Your Python**

Y t t f th l f th t bl b H ’ l

https://realpython.com/linear-regression-in-python/ 17/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

You can extract any of the values from the table above. Here’s an example:

Python >>>

>>> print('coefficient of determination:', results.rsquared)

coefficient of determination: 0.8615939258756777

>>> print('adjusted coefficient of determination:', results.rsquared\_adj)

adjusted coefficient of determination: 0.8062314962259488

>>> print('regression coefficients:', results.params)

regression coefficients: [5.52257928 0.44706965 0.25502548]

That’s how you obtain some of the results of linear regression:

1. **.rsquared** holds ��².

**Improve Your Python**

2. **.rsquared\_adj** represents adjusted ��² (��² corrected according to the number of input features). ...with a fresh �� **Python Trick** ��

3. **.params** refers the array with ��₀, ��₁, and ��₂respectively.

code snippet every couple of days: Email Address

You can also notice that these results are identical to those obtained with scikit-learn for the same problem.

To find more information about the results of linear regression, please visit the official documentation page. **Send Python Tricks »**

**Step 5: Predict response**

You can obtain the predicted response on the input values used for creating the model using .fittedvalues or .predict() with the input array as the argument:

Python >>>

>>> print('predicted response:', results.fittedvalues, sep='\n')

predicted response:

[ 5.77760476 8.012953 12.73867497 17.9744479 23.97529728 29.4660957

38.78227633 41.27265006]

>>> print('predicted response:', results.predict(x), sep='\n')

predicted response:

[ 5.77760476 8.012953 12.73867497 17.9744479 23.97529728 29.4660957

38.78227633 41.27265006]

This is the predicted response for known inputs. If you want predictions with new regressors, you can also apply .predict() with new data as the argument:

Python >>>

>>> x\_new = sm.add\_constant(np.arange(10).reshape((-1, 2)))

>>> print(x\_new)

[[1. 0. 1.]

[1. 2. 3.]

[1. 4. 5.]

[1. 6. 7.]

[1. 8. 9.]]

>>> y\_new = results.predict(x\_new)

>>> print(y\_new)

[ 5.77760476 7.18179502 8.58598528 9.99017554 11.3943658 ]

You can notice that the predicted results are the same as those obtained with scikit-learn for the same problem.  Remove ads

**Beyond Linear Regression**

Linear regression is sometimes not appropriate, especially for non-linear models of high complexity.

Fortunately, there are other regression techniques suitable for the cases where linear regression doesn’t work well. Some of them are support vector machines, decision trees, random forest, and neural networks. **Improve Your Python**

https://realpython.com/linear-regression-in-python/ 18/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

There are numerous Python libraries for regression using these techniques. Most of them are free and open-source. That’s one of the reasons why Python is among the main programming languages for machine learning.

The package scikit-learn provides the means for using other regression techniques in a very similar way to what you’ve seen. It contains the classes for support vector machines, decision trees, random forest, and more, with the methods .fit(), .predict(), .score() and so on.

**Conclusion**

You now know what linear regression is and how you can implement it with Python and three open-source packages:

NumPy, scikit-learn, and statsmodels. 

You use NumPy for handling arrays.

Linear regression is implemented with the following:

**Improve Your Python**

...with a fresh �� **Python Trick** �� code snippet every couple of days:

**scikit-learn** if you don’t need detailed results and want to use the approach consistent with other regression

techniques

**statsmodels** if you need the advanced statistical parameters of a model

Email Address

**Send Python Tricks »**

Both approaches are worth learning how to use and exploring further. The links in this article can be very useful for that.

When performing linear regression in Python, you can follow these steps:

1. Import the packages and classes you need

2. Provide data to work with and eventually do appropriate transformations

3. Create a regression model and fit it with existing data

4. Check the results of model fitting to know whether the model is satisfactory

5. Apply the model for predictions

If you have questions or comments, please put them in the comment section below.

Mark as Completed 

�� Python Tricks ��

Get a short & sweet **Python Trick** delivered to your inbox every couple of 

days. No spam ever. Unsubscribe any time. Curated by the Real Python

team.

Email Address

Send Me Python Tricks »

About **Mirko Stojiljkovi**ć

Mirko has a Ph.D. in Mechanical Engineering and works as a university professor. He is a Pythonis

who applies hybrid optimization and machine learning methods to support decision making in th

**Improve Your Python**

https://realpython.com/linear-regression-in-python/ 19/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

energy sector.

» More about Mirko

*Each tutorial at Real Python is created by a team of developers so that it meets our high quality standards. The team members who worked on this tutorial are:*

**Improve Your Python** 

Aldren Joanna Kyle

...with a fresh �� **Python Trick** ��

code snippet every couple of days:

Email Address

Master Real-World Python Skills

**~~Se~~nd Python Tricks »**

With Unlimited Access to Real Python

**Join us and get access to hundreds of tutorials, hands**

**on video courses, and a community of expert**

**Pythonistas:**

Level Up Your Python Skills »

What Do You Think?

 **Tweet**  **Share**  **Email**

**Real Python Comment Policy:** The most useful comments are those written with the goal of learning from or helping out other readers—after reading the whole article and all the earlier comments.

Complaints and insults generally won’t make the cut here.

What’s your #1 takeaway or favorite thing you learned? How are you going to put your newfound skills to use? Leave a comment below and let us know.

Keep Learning

Related Tutorial Categories: **data-science intermediate machine-learning**

**Improve Your Python**

— FREE Email Series —

https://realpython.com/linear-regression-in-python/ 20/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

�� Python Tricks ��



Email… 

Get Python Tricks »

�� No spam. Unsubscribe any time.

All Tutorial Topics

**Improve Your Python**

...with a fresh �� **Python Trick** �� code snippet every couple of days:

Email Address

**Send Python Tricks »**

**advanced api basics best-practices community databases data-science devops django docker flask front-end gamedev gui intermediate machine-learning projects python testing tools web-dev web-scraping**

****

Table of Contents

Regression

Linear Regression

→ Implementing Linear Regression in Python

Beyond Linear Regression

Conclusion

Mark as Completed 

**Improve Your Python**

 **Tweet**  **Share**  **Email**

https://realpython.com/linear-regression-in-python/ 21/22

20/12/2021, 13:30 Linear Regression in Python – Real Python

 **Tweet**  **Share**  **Email**

****

**Improve Your Python** 

 Remove ads

...with a fresh �� **Python Trick** ��

© 2012–2021 Real Python ⋅ Newsletter ⋅ Podcast ⋅ YouTube ⋅ Twitter ⋅ Facebook ⋅ Instagram ⋅ co~~de snippe~~t e~~very coup~~le of days:

Python Tutorials ⋅ Search ⋅ Privacy Policy ⋅ Energy Policy ⋅ Advertise ⋅ Contact

❤️ Happy Pythoning!

Email Address

**Send Python Tricks »**

**Improve Your Python**

https://realpython.com/linear-regression-in-python/ 22/22