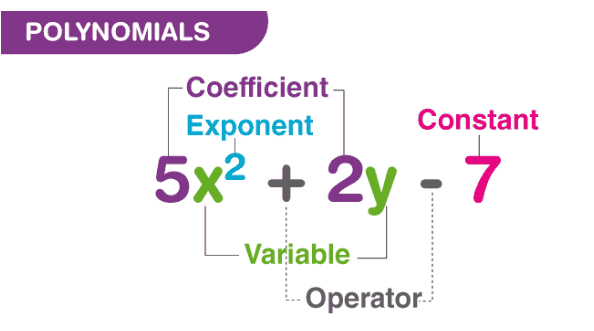
**Polynomial**

**Polynomials** are algebraic expressions that consist of variables and coefficients. Variables are also sometimes called indeterminate. We can perform [arithmetic operations](https://byjus.com/maths/arithmetic-operations/) such as addition, subtraction, multiplication and also positive integer exponents for polynomial expressions but not division by variable. An example of a polynomial with one variable is x2+x-12. In this example, there are three terms: x2, x and -12.

The word polynomial is derived from the Greek words ‘poly’ means ‘**many**‘ and ‘nominal’ means ‘**terms**‘, so altogether it said “many terms”. A polynomial can have any number of terms but not infinite. Learn about degree, terms, types, properties, polynomial functions



Polynomial is made up of two terms, namely Poly (meaning “many”) and Nominal (meaning “terms.”). A polynomial is defined as an expression which is composed of variables, constants and exponents, that are combined using the mathematical operations such as addition, subtraction, multiplication and division (No division operation by a variable). Based on the numbers of terms present in the expression, it is classified as monomial, binomial, and trinomial. Examples of constants, variables and exponents are as follows:

* Constants. Example: 1, 2, 3, etc.
* Variables. Example: g, h, x, y, etc.
* Exponents: Example: 5 in x5 etc.

### Notation

The polynomial function is denoted by P(x) where x represents the variable. For example,

P(x) = x2-5x+11

If the variable is denoted by a, then the function will be P(a)

## Degree of a Polynomial

The [degree of a polynomial](https://byjus.com/maths/degree-of-a-polynomial/) is defined as the highest degree of a monomial within a polynomial. Thus, a polynomial equation having one variable which has the largest exponent is called a degree of the polynomial.

| **Polynomial** | **Degree** | **Example** |
| --- | --- | --- |
| Constant or Zero Polynomial | 0 | c |
| Linear Polynomial | 1 | ax+b |
| Quadratic Polynomial | 2 | ax2+bx+c |
| Cubic Polynomial | 3 | ax3+bx3+cx+d |
| Quartic Polynomial | 4 | ax4+bx3+cx2+dx+e |
| Quintic Polynomial | 5 | ax5+bx4+cx3+dx2+ex+f |

## 

The polynomial model can go up to the sixth degree. A larger magnitude corresponds to a greater adjustment than that in the original data; however, this does not mean that it is best for forecasting. The best method is the one that can perform well with minimum parameters.

## 

## Linear Polynomials

A **linear polynomial** is a polynomial of degree **one,**i.e., the highest [exponent](https://www.cuemath.com/algebra/exponents/) of the variable is one, defined by an [equation](https://www.cuemath.com/algebra/equation/) of the form: p(x): ax + b, a≠0.

We note that a linear polynomial in one variable can have at the most two terms. The constraint that a should not be equal to 0 is required because if a is 0, then this becomes a constant polynomial.

## Quadratic Polynomials

A **quadratic polynomial** is a polynomial of degree **two ,**i.e., the highest exponent of the variable is two. In general, a quadratic polynomial will be of the form: p(x): ax2 + bx + c, a≠0

## We observe that a quadratic polynomial can have at the most three terms. The constraint that a should not be equal to 0 is required because if a is 0, then this becomes a linear polynomial.

## Cubic Polynomials

A **cubic polynomial** is a polynomial of degree three, i.e., the highest exponent of the variable is three. A cubic polynomial, in general, will be of the form p(x): ax3 + bx2 + cx + d, a≠0

We observe that a cubic polynomial can have at the most four terms. Once again, the constraint that a should not be equal to 0 is required because if a is 0, then this becomes a quadratic rather than a cubic polynomial.

## Terms of a Polynomial

The terms of polynomials are the parts of the equation which are generally separated by “+” or “-” signs. So, each part of a polynomial in an equation is a term. For example, in a polynomial, say, 2x2 + 5 +4, the number of terms will be 3. The classification of a polynomial is done based on the number of terms in it.

|  |  |  |
| --- | --- | --- |
| **Polynomial** | **Terms** | **Degree** |
| P(x) = x3-2x2+3x+4 | x3, -2x2, 3x and 4 | 3 |

## Types of Polynomials

Polynomials are of 3 different types and are classified based on the number of terms in it. The three types of polynomials are:

* **Monomial**
* **Binomial**
* **Trinomial**

These polynomials can be combined using addition, subtraction, multiplication, and division but is never division by a variable. A few examples of **Non Polynomials** are: 1/x+2, x-3

### Monomial

A monomial is an expression which contains only one term. For an expression to be a monomial, the single term should be a non-zero term. A few examples of monomials are:

* 5x
* 3
* 6a4
* -3xy

### Binomial

A binomial is a polynomial expression which contains exactly two terms. A binomial can be considered as a sum or difference between two or more monomials. A few examples of binomials are:

* – 5x+3,
* 6a4 + 17x
* xy2+xy

### Trinomial

A trinomial is an expression which is composed of exactly three terms. A few examples of trinomial expressions are:

* – 8a4+2x+7
* 4x2 + 9x + 7

|  |  |  |
| --- | --- | --- |
| **Monomial** | **Binomial** | **Trinomial** |
| One Term | Two terms | Three terms |
| Example: x, 3y, 29, x/2 | Example: x2+x, x3-2x, y+2 | Example: x2+2x+20 |

NUMPY.POLYFIT

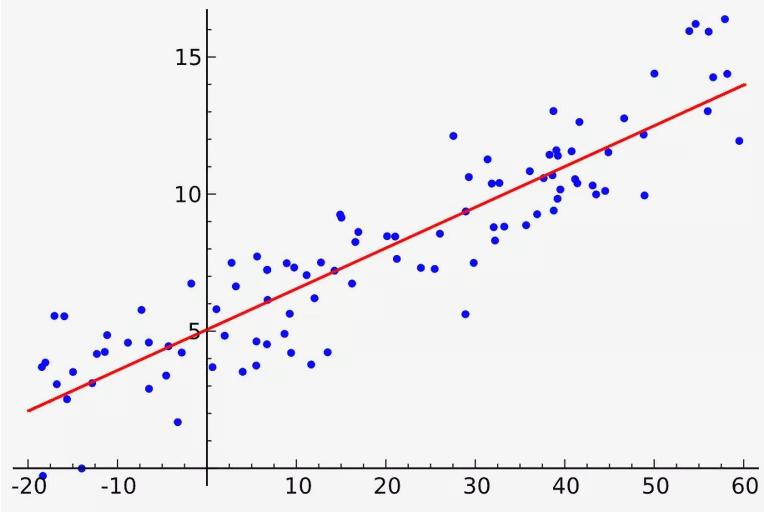
**numpy.polyfit(x, y, deg, rcond=None, full=False, w=None, cov=False)**

Least squares polynomial fit.

**It used the ordinary least squares method (which is often referred to with its short form: OLS)**

When you fit a line to your dataset, for most x values there is a difference between the y value that your model estimates — and the real y value that you have in your dataset. In machine learning, this difference is called **error**.

The least-squares method is a mathematical technique that allows the analyst to determine the best way of fitting a curve on top of a chart of data points. It is widely used to make scatter plots easier to interpret and is associated with [regression analysis](https://www.investopedia.com/terms/r/regression.asp). These days, the least-squares method can be used as part of most statistical software programs.



**Least squares Img**

## 

## Calculation for slope and intercept

## Line of Best Fit

Since the least squares line **minimizes the squared distances** between the line and our points, we can think of this line as the one that best fits our data. This is why the **least squares line** is also known as the **line of best fit**. Of all of the possible lines that could be drawn, the least squares line **is closest to the set of data as a whole**. This may mean that our line will miss hitting any of the points in our set of data.

Fit a polynomial p(x) = p[0] \* x\*\*deg + ... + p[deg] of degree deg to points (x, y). Returns a vector of coefficients p that minimizes the squared error in the order deg, deg-1, … 0.

**Parameters :**

**xarray\_like, shape (M,)**

X-coordinates of the M sample points (x[i], y[i]).

**yarray\_like, shape (M,) or (M, K)**

Y-coordinates of the sample points. Several data sets of sample points sharing the same x-coordinates can be fitted at once by passing in a 2D-array that contains one dataset per column.

**degint**

Degree of the fitting polynomial

**rcondfloat, optional**

Relative condition number of the fit. Singular values smaller than this relative to the largest singular value will be ignored. The default value is len(x)\*eps, where eps is the relative precision of the float type, about 2e-16 in most cases.

**fullbool, optional**

Switch determining nature of return value. When it is False (the default) just the coefficients are returned, when True diagnostic information from the singular value decomposition is also returned.

**warray\_like, shape (M,), optional**

Weights to apply to the y-coordinates of the sample points. For Gaussian uncertainties, use 1/sigma (not 1/sigma\*\*2)

**covbool or str, optional**

If given and not False, return not just the estimate but also its covariance matrix. By default, the covariance are scaled by chi2/dof, where dof = M - (deg + 1), i.e., the weights are presumed to be unreliable except in a relative sense and everything is scaled such that the reduced chi2 is unity. This scaling is omitted if cov='unscaled', as is relevant for the case that the weights are 1/sigma\*\*2, with sigma known to be a reliable estimate of the uncertainty.

**Returns:**

**pndarray, shape (deg + 1,) or (deg + 1, K)**

Polynomial coefficients, highest power first. If y was 2-D, the coefficients for k-th data set are in p[:,k].

**residuals, rank, singular\_values, rcond**

Present only if [**full**](https://numpy.org/doc/stable/reference/generated/numpy.full.html#numpy.full) = True. Residuals is sum of squared residuals of the least-squares fit, the effective rank of the scaled Vandermonde coefficient matrix, its singular values, and the specified value of rcond. For more details, see [**linalg.lstsq**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html#numpy.linalg.lstsq).

**Vndarray, shape (M,M) or (M,M,K)**

Present only if [**full**](https://numpy.org/doc/stable/reference/generated/numpy.full.html#numpy.full) = False and cov`=True. The covariance matrix of the polynomial coefficient estimates. The diagonal of this matrix are the variance estimates for each coefficient. If y is a 2-D array, then the covariance matrix for the `k-th data set are in V[:,:,k]

**Warns :**

**RankWarning**

The rank of the coefficient matrix in the least-squares fit is deficient. The warning is only raised if [**full**](https://numpy.org/doc/stable/reference/generated/numpy.full.html#numpy.full) = False.

The warnings can be turned off by

>>> **import** warnings

>>> warnings**.**simplefilter**(**'ignore'**,** np**.**RankWarning**)**

**Notes**

The solution minimizes the squared error

E=∑j=0k|p(xj)−yj|2

in the equations:

x**[0]\*\***n **\*** p**[0]** **+** **...** **+** x**[0]** **\*** p**[**n**-1]** **+** p**[**n**]** **=** y**[0]**

x**[1]\*\***n **\*** p**[0]** **+** **...** **+** x**[1]** **\*** p**[**n**-1]** **+** p**[**n**]** **=** y**[1]**

**...**

x**[**k**]\*\***n **\*** p**[0]** **+** **...** **+** x**[**k**]** **\*** p**[**n**-1]** **+** p**[**n**]** **=** y**[**k**]**

[**polyfit**](https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html#numpy.polyfit) issues a [**RankWarning**](https://numpy.org/doc/stable/reference/generated/numpy.RankWarning.html#numpy.RankWarning) when the least-squares fit is badly conditioned. This implies that the best fit is not well-defined due to numerical error. The results may be improved by lowering the polynomial degree or by replacing x by x - x.mean(). The rcond parameter can also be set to a value smaller than its default, but the resulting fit may be spurious: including contributions from the small singular values can add numerical noise to the result.

Note that fitting polynomial coefficients is inherently badly conditioned when the degree of the polynomial is large or the interval of sample points is badly centered. The quality of the fit should always be checked in these cases. When polynomial fits are not satisfactory, splines may be a good alternative.

**numpy.poly1d() in Python**

(poly1d. A **one-dimensional polynomial class)**

**Equation:** based on degree 3 is ax3+bx2+cx+d= y , degree 2 is ax2+bx+c=y, degree 1 is ax+b=y.

Where a is not equal 0 for all the 3 degree

p **=** np**.**poly1d **([1,** **2,** **3])**

>>> print**(**np**.**poly1d**(**p**))**

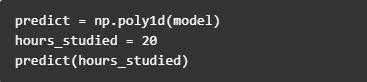
ax3+bx2+cx+d

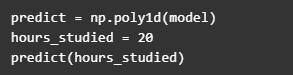
Evaluate the polynomial at x=value (0.5)

Output: based on the equation

We finally got our equation that describes the fitted line.

Eg. (If a student tells you how many hours she studied, you can predict the estimated results of her exam)





Predict = np.poly1d(model)

Study = 20

Predict (Study)

Output : You can use these coefficient and intercept values – and the poly1d() method – to estimate unknown values.

The **numpy.poly1d()** function helps to define a polynomial function. It makes it easy to apply “natural operations” on polynomials.

**Syntax:**numpy.poly1d(arr, root, var)

**Parameters:**

**arr:** [array\_like] The polynomial coefficients are given in decreasing order of powers. If the second parameter (root) is set to True then array values are the roots of the polynomial equation.

**root:** [bool, optional] True means polynomial roots. Default is False.

**var:** variable like x, y, z that we need in polynomial [default is x].

**Arguments:**

**c :** Polynomial coefficient.

**coef :** Polynomial coefficient.

**Coefficients:** Polynomial coefficient.

**Order:** Order or degree of polynomial.

**o:** Order or degree of polynomial.

**r:** Polynomial root.

**roots:** Polynomial root.

**Return:** Polynomial and the operation applied

**And this is how you do predictions by using machine learning and simple linear regression in Python.**

**ACCURACY METRICS**

There are a few methods to calculate **the accuracy of your model.**

The **R-squared (R2) value .**

The R-squared value is a number between 0 and 1. And the closer it is to 1 the more accurate your linear regression model is.

**R-squared calculation is not implemented in numpy so** that one should be borrowed from sklearn.





And now we know our R-squared value is if near to 1 model is well and good if near to 0 its worst.