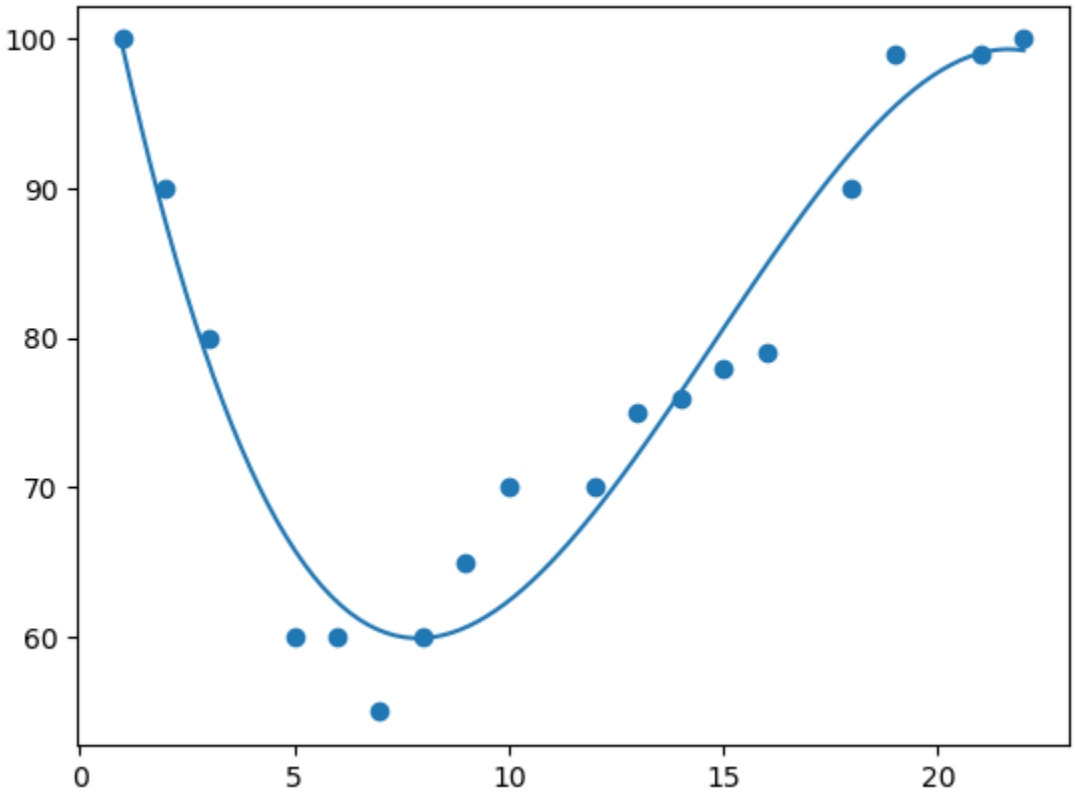
Machine Learning - Polynomial Regression

Polynomial Regression

If your data points clearly will not fit a linear regression (a straight line through all data points), it might be ideal for polynomial regression.

Polynomial regression, like linear regression, uses the relationship between the variables x and y to find the best way to draw a line through the data points.



How Does it Work?

Python has methods for finding a relationship between data-points and to draw a line of polynomial regression. We will show you how to use these methods instead of going through the mathematic formula.

In the example below, we have registered 18 cars as they were passing a certain tollbooth.

We have registered the car's speed, and the time of day (hour) the passing occurred.

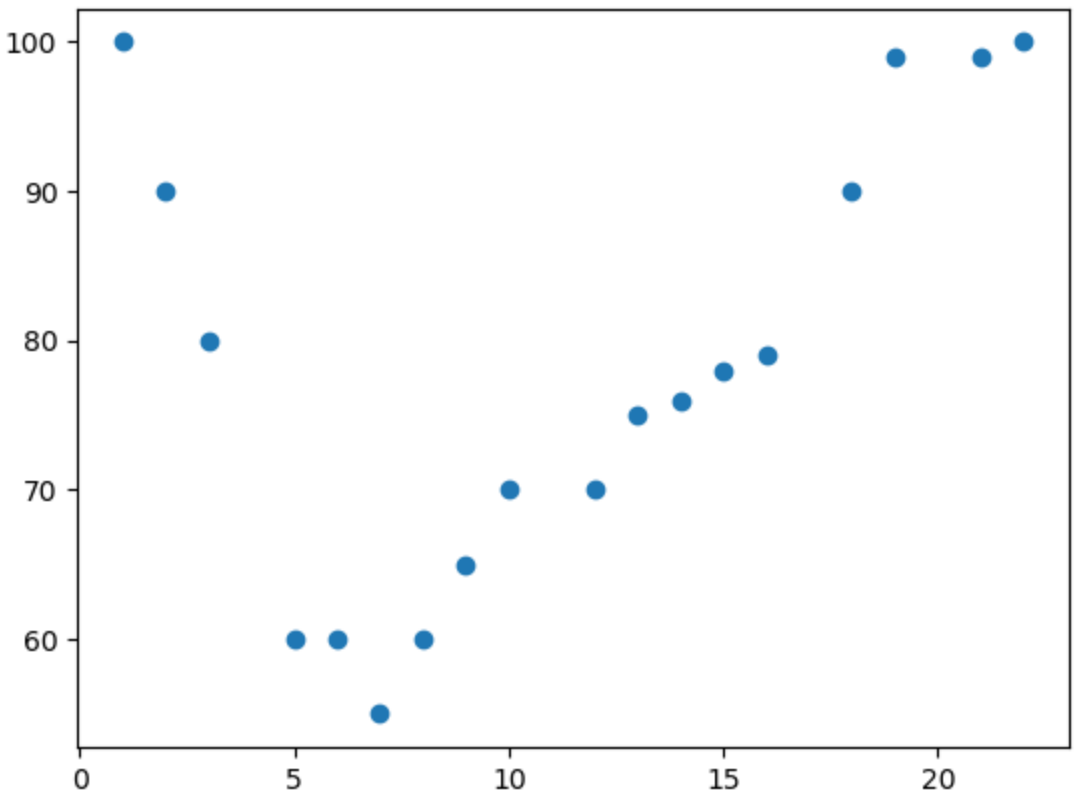
The x-axis represents the hours of the day and the y-axis represents the speed:

Example

Start by drawing a scatter plot:

```
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

plt.scatter(x, y)
plt.show()
```

Result:

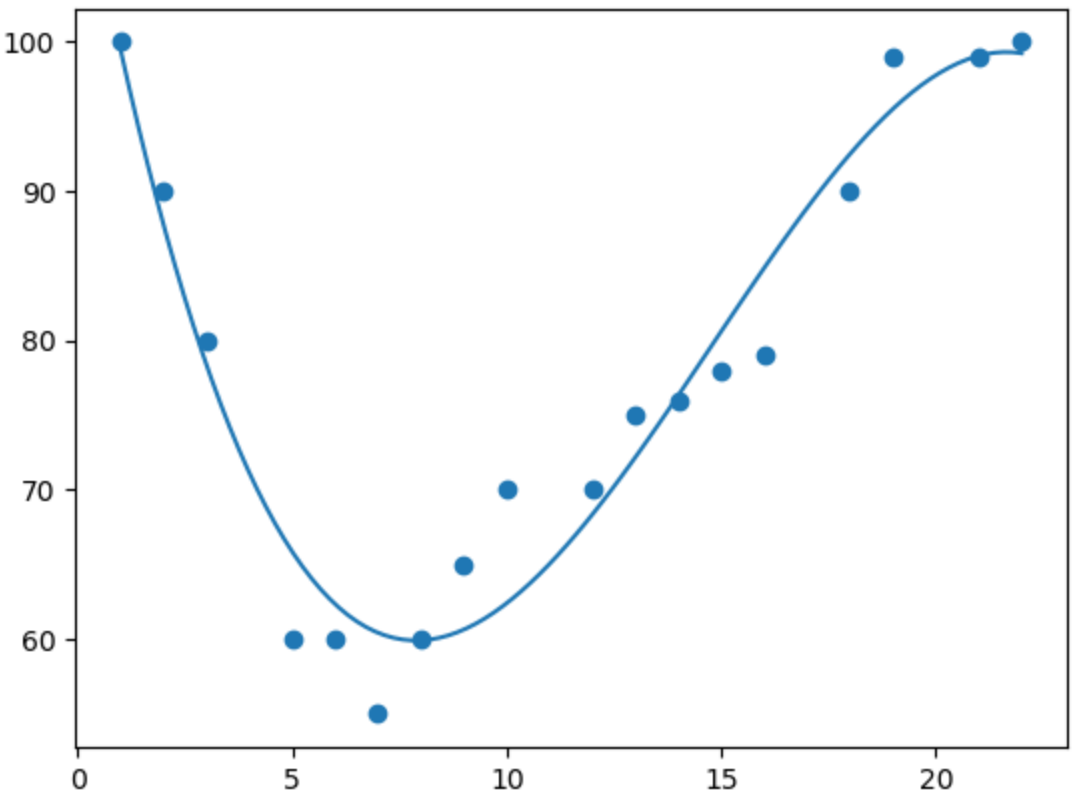


Example

Import `numpy` and `matplotlib` then draw the line of Polynomial Regression:

```
import numpy
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

myline = numpy.linspace(1, 22, 100)

plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Result:



Example Explained

Import the modules you need.

You can learn about the NumPy module in our NumPy Tutorial.

You can learn about the SciPy module in our SciPy Tutorial.

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

Create the arrays that represent the values of the x and y axis:

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

NumPy has a method that lets us make a polynomial model:

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

Then specify how the line will display, we start at position 1, and end at position 22:

```
myline = numpy.linspace(1, 22, 100)
```

Draw the original scatter plot:

```
plt.scatter(x, y)
```

Draw the line of polynomial regression:

```
plt.plot(myline, mymodel(myline))
```

Display the diagram:

```
plt.show()
```

R-Squared

It is important to know how well the relationship between the values of the x- and y-axis is, if there are no relationship the polynomial regression can not be used to predict anything.

The relationship is measured with a value called the r-squared.

The r-squared value ranges from 0 to 1, where 0 means no relationship, and 1 means 100% related.

Python and the Sklearn module will compute this value for you, all you have to do is feed it with the x and y arrays:

Example

How well does my data fit in a polynomial regression?

```
import numpy
from sklearn.metrics import r2_score
```

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

print(r2_score(y, mymodel(x)))
```

Note: The result 0.94 shows that there is a very good relationship, and we can use polynomial regression in future predictions.

Predict Future Values

Now we can use the information we have gathered to predict future values.

Example: Let us try to predict the speed of a car that passes the tollbooth at around 17 P.M:

To do so, we need the same `mymodel` array from the example above:

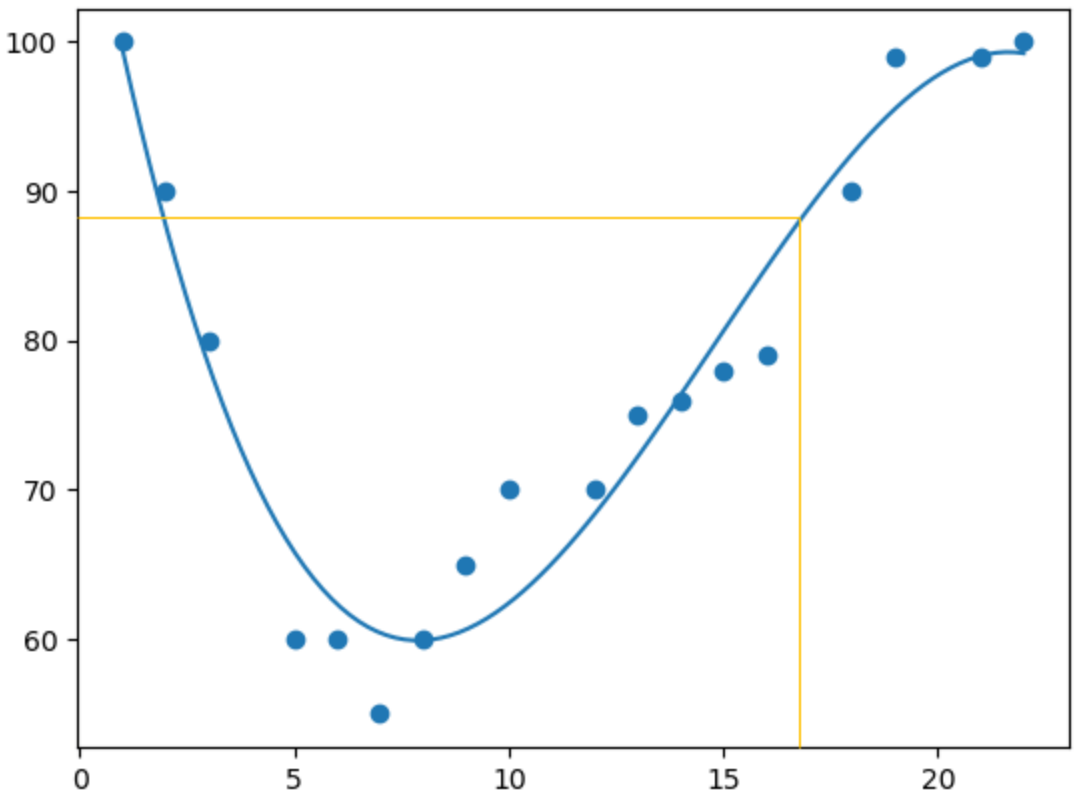
```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

Example

Predict the speed of a car passing at 17 P.M:

```
import numpy
from sklearn.metrics import r2_score

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

speed = mymodel(17)
print(speed)
```

The example predicted a speed to be 88.87, which we also could read from the diagram:



Bad Fit?

Let us create an example where polynomial regression would not be the best method to predict future values.

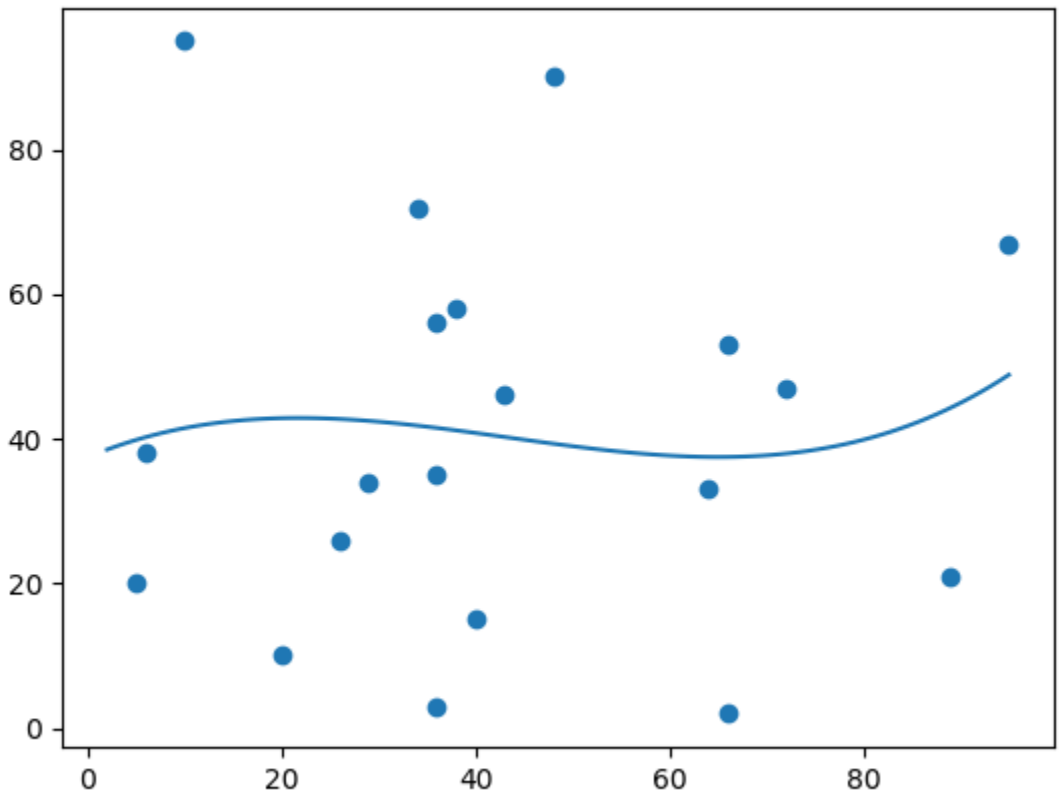
Example

These values for the x- and y-axis should result in a very bad fit for polynomial regression:

```
import numpy
import matplotlib.pyplot as plt

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
```

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

myline = numpy.linspace(2, 95, 100)

plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Result:



And the r-squared value?

Example

You should get a very low r-squared value.

```
import numpy
from sklearn.metrics import r2_score
```

```
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

print(r2_score(y, mymodel(x)))
```

Machine Learning - Scale

Scale Features

When your data has different values, and even different measurement units, it can be difficult to compare them. What is kilograms compared to meters? Or altitude compared to time?

The answer to this problem is scaling. We can scale data into new values that are easier to compare.

Take a look at the table below, it is the same data set that we used in the multiple regression chapter, but this time the **volume** column contains values in *liters* instead of cm^3 (1.0 instead of 1000).

The file is meant for testing purposes only, you can download it here: cars2.csv

Car	Model	Volume	Weight
Toyota	Aygo	1.0	790
Mitsubishi	Space Star	1.2	1160
Skoda	Citigo	1.0	929
Fiat	500	0.9	865
Mini	Cooper	1.5	1140

VW	Up!	1.0	929
Skoda	Fabia	1.4	1109
Mercedes	A-Class	1.5	1365
Ford	Fiesta	1.5	1112
Audi	A1	1.6	1150
Hyundai	I20	1.1	980
Suzuki	Swift	1.3	990
Ford	Fiesta	1.0	1112
Honda	Civic	1.6	1252
Hundai	I30	1.6	1326
Opel	Astra	1.6	1330
BMW	1	1.6	1365
Mazda	3	2.2	1280
Skoda	Rapid	1.6	1119
Ford	Focus	2.0	1328
Ford	Mondeo	1.6	1584
Opel	Insignia	2.0	1428
Mercedes	C-Class	2.1	1365
Skoda	Octavia	1.6	1415
Volvo	S60	2.0	1415
Mercedes	CLA	1.5	1465
Audi	A4	2.0	1490
Audi	A6	2.0	1725
Volvo	V70	1.6	1523
BMW	5	2.0	1705

Mercedes	E-Class	2.1	1605
Volvo	XC70	2.0	1746
Ford	B-Max	1.6	1235
BMW	2	1.6	1390
Opel	Zafira	1.6	1405
Mercedes	SLK	2.5	1395

It can be difficult to compare the volume 1.0 with the weight 790, but if we scale them both into comparable values, we can easily see how much one value is compared to the other.

There are different methods for scaling data, in this tutorial we will use a method called standardization.

The standardization method uses this formula:

```
z = (x - u) / s
```

Where `z` is the new value, `x` is the original value, `u` is the mean and `s` is the standard deviation.

If you take the **weight** column from the data set above, the first value is 790, and the scaled value will be:

```
(790 - 1292.23) / 238.74 = -2.1
```

If you take the **volume** column from the data set above, the first value is 1.0, and the scaled value will be:

```
(1.0 - 1.61) / 0.38 = -1.59
```

Now you can compare -2.1 with -1.59 instead of comparing 790 with 1.0.

You do not have to do this manually, the Python sklearn module has a method called `StandardScaler()` which returns a Scaler object with methods for transforming data sets.

Example

Scale all values in the Weight and Volume columns:

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()

df = pandas.read_csv("cars2.csv")

X = df[['Weight', 'Volume']]

scaledX = scale.fit_transform(X)

print(scaledX)
```

Result:

Note that the first two values are -2.1 and -1.59, which corresponds to our calculations:

```
[[-2.10389253 -1.59336644]
 [-0.55407235 -1.07190106]
 [-1.52166278 -1.59336644]
 [-1.78973979 -1.85409913]
 [-0.63784641 -0.28970299]
 [-1.52166278 -1.59336644]
 [-0.76769621 -0.55043568]
 [ 0.3046118  -0.28970299]
 [-0.7551301  -0.28970299]
 [-0.59595938 -0.0289703 ]
 [-1.30803892 -1.33263375]
 [-1.26615189 -0.81116837]
 [-0.7551301  -1.59336644]
 [-0.16871166 -0.0289703 ]
 [ 0.14125238 -0.0289703 ]
 [ 0.15800719 -0.0289703 ]
 [ 0.3046118  -0.0289703 ]
 [-0.05142797  1.53542584]
 [-0.72580918 -0.0289703 ]
 [ 0.14962979  1.01396046]
 [ 1.2219378  -0.0289703 ]
 [ 0.5685001   1.01396046]
 [ 0.3046118   1.27469315]
 [ 0.51404696 -0.0289703 ]
 [ 0.51404696  1.01396046]
 [ 0.72348212 -0.28970299]
 [ 0.8281997   1.01396046]
 [ 1.81254495  1.01396046]
 [ 0.96642691 -0.0289703 ]
 [ 1.72877089  1.01396046]
```

```
 [ 1.30990057  1.27469315]
 [ 1.90050772  1.01396046]
 [-0.23991961 -0.0289703 ]
 [ 0.40932938 -0.0289703 ]
 [ 0.47215993 -0.0289703 ]
 [ 0.4302729   2.31762392]]
```

Predict CO2 Values

The task in the Multiple Regression chapter was to predict the CO2 emission from a car when you only knew its weight and volume.

When the data set is scaled, you will have to use the scale when you predict values:

Example

Predict the CO2 emission from a 1.3 liter car that weighs 2300 kilograms:

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()

df = pandas.read_csv("cars2.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

scaledX = scale.fit_transform(X)

regr = linear_model.LinearRegression()
regr.fit(scaledX, y)

scaled = scale.transform([[2300, 1.3]])

predictedCO2 = regr.predict([scaled[0]])
print(predictedCO2)
```

Result:

```
[107.2087328]
```