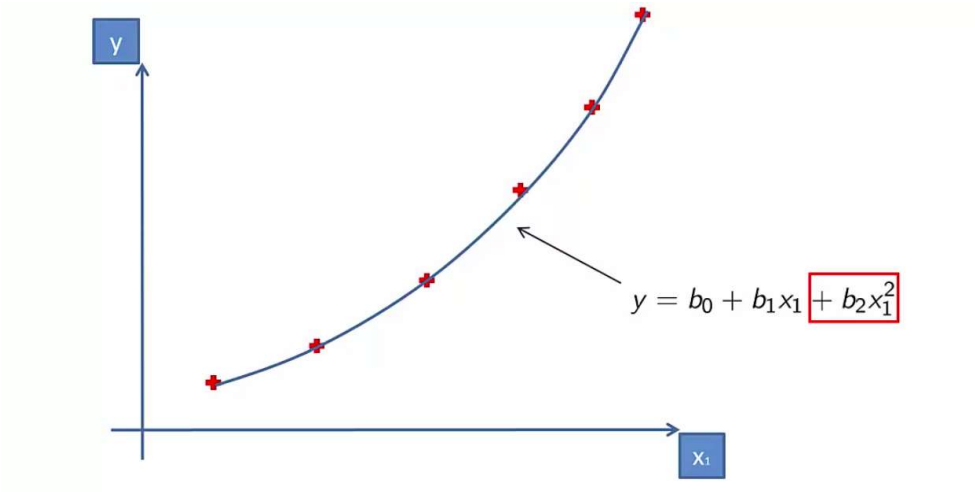
Module 2-2: Polynomial Regression

Linear regression is great and all, but sometimes data just doesn't come out that way. Often times your data won't fit neatly into a line, and you'll end up getting very inaccurate results if you based your data on one line.

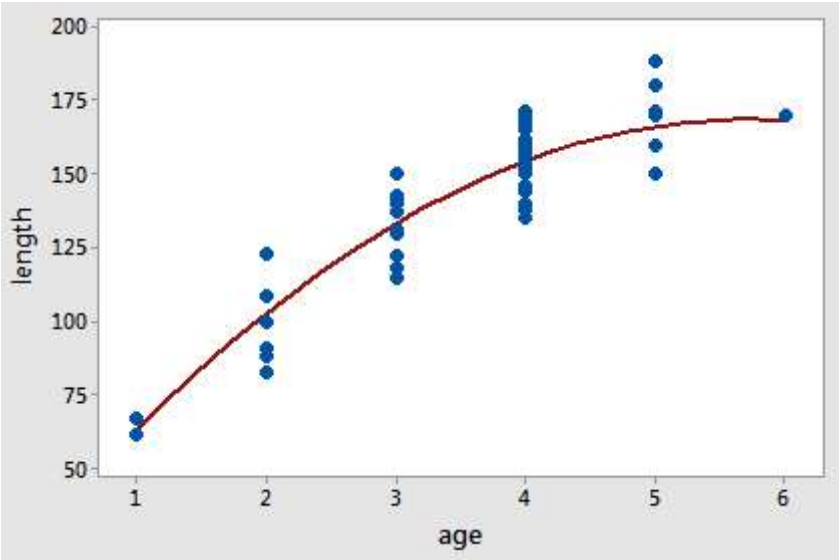
Similar to linear regression, **polynomial regression** finds the relationship between the x and y points in a graph. However, in polynomial regression, the pattern that is identified would not be a straight line, but a curve instead. More accurately, the curve is made of many straight lines put together.

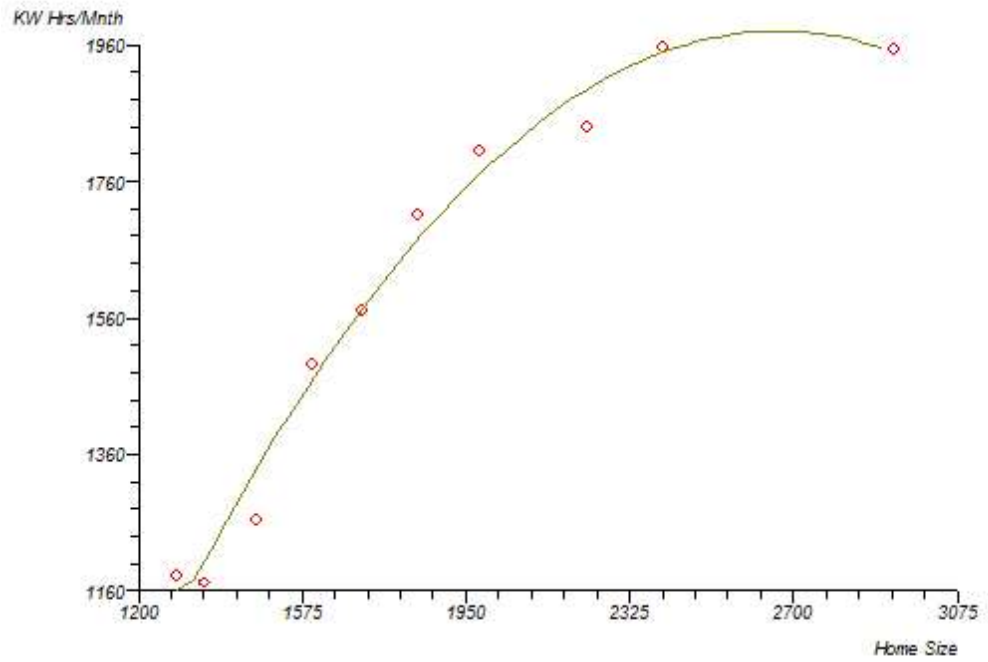


Taking the previous example of the ice cream sales, a polynomial regression relationship would be similar to this curve.

Polynomial regression can be fit to many different types of data. Some examples of data that can fit to a curve include tracking the age of an animal based on its size and electricity usage based on home size.

Send Issue





In linear regression, the equation for a line is:

Text COPY

```
y = mx + b
```

However, in polynomial regression, the degree of the equation can be any number greater than 1. For example, these are two possible polynomial regression equations:

Text COPY

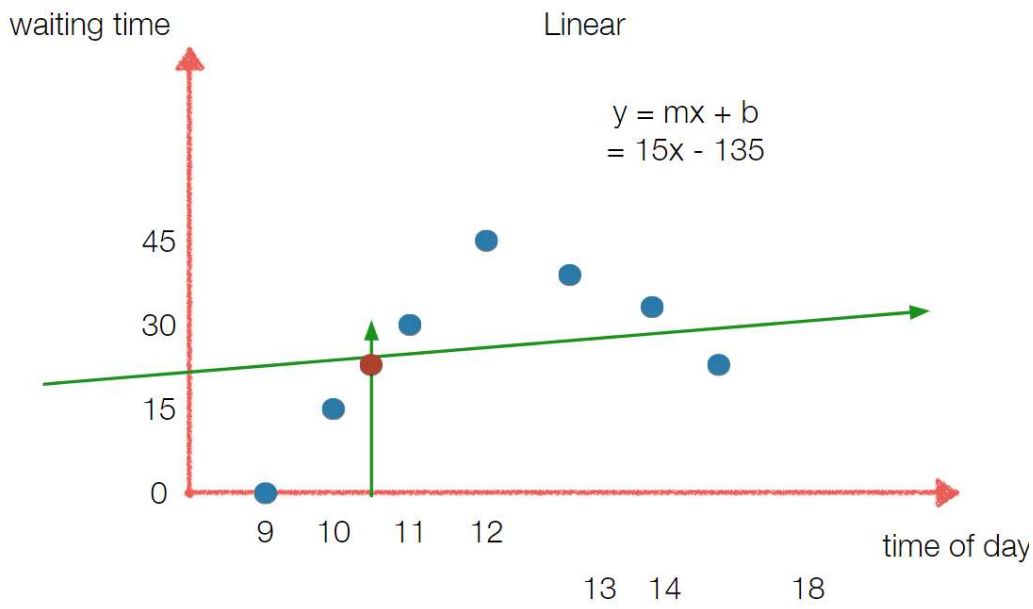
```
y = ax^2 + bx + c

y = ax^3 + bx^2 + cx + d
```

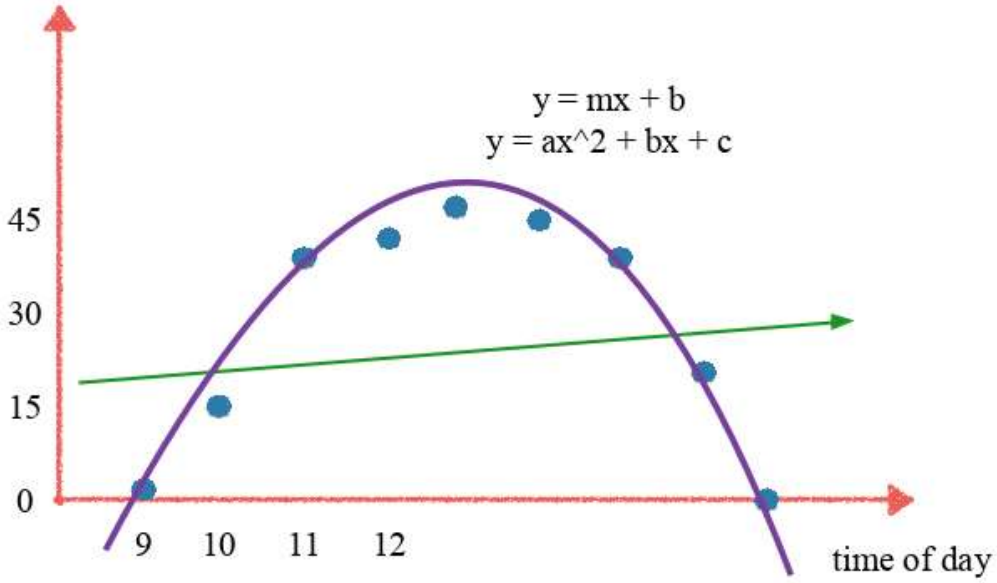
The degree of these equations are the highest exponent of the variable `x`, which has a degree of 2 and 3 respectively in the equations above. `a`, `b`, `c`, and `d` can be any number.

Using Polynomial Regression

You might have noticed that in our previous lesson the data for the time spent waiting in line at Disneyland didn't really fit a line at all. Like, it's **really** obvious. If you follow the line of best fit created using linear regression, the higher the time of day gets the longer the waiting time will be.



Thankfully we know how to fix it. Polynomial regression is a better option because it allows us to create a curve that can more accurately represent the data.



In the program we will be doing today, we will continue with the previous Disneyland problem and utilize polynomial regression instead of linear regression.

Before starting on the polynomial regression program, to review from last class, the steps for creating most artificial intelligence programs include:

1. Define your data table
2. Collect your data
3. Choose a model
4. Machine Learning (fit)
5. Prediction

The first two steps is to define your data table and collect your data. We will be using the data collected from last week.

Time of the Day	Waiting Time
9	0
10	10
11	20
12	30
13	40
14	30
15	20
16	10

In this graph, the green line is linear regression and the purple line is polynomial regression. It is very clear that the polynomial regression line is a much better match to represent the data points given.

Now we will begin creating our polynomial regression program based on this data. The first two steps of defining your data table and collecting your data will be exactly like last week's linear regression program.

Just like before, you will need to begin with importing the `sklearn` library and other necessary imports.

Python COPY

```
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
```

The next part is to create your input and output from the table that was created before. The input will be in the form of a list within a list and the output will also be a list.

Python COPY

```
input_data = [
    [9],
    [10],
    [11],
    [12],
    [13],
    [14],
    [15],
    [16],
]

output_data = [0,10,20,30,40,30,20,10]
```

The next step of machine learning is where there is a difference in the program because instead of a linear model, we will be using the polynomial model. The polynomial regression model is strikingly similar to the linear regression model, but with a few key differences.

Python COPY

```
model = make_pipeline(PolynomialFeatures(2), Ridge())
model.fit(input_data, output_data)
```

```
print(model.predict([[11.5]]))
```

In the first line of code, we create our polynomial regression model based on sklearn's `PolynomialFeatures`, but we use what is known as a **pipeline** to help handle some of the more difficult tasks that come with polynomial regression. Unlike a linear model, we cannot simply fit the input and output into the polynomial model. The pipeline takes in the transformed input data and fits it as polynomial input for the input and output data of the model.

The next difference to our code is that we provided a degree for the polynomial features. Just as we showed with our discussion on how to calculate polynomial regression mathematically, we need to provide a degree for our formula; with most parabolas like this one it is just a degree of 2. This basically works out to the number of changes in direction for our line.

Lastly, we added the `Ridge()` parameter, which lets the program know how we are going to handle our polynomial regression.

This will create the polynomial regressive model according to the transformed data from the `PolynomialFeatures`.

In all, at 11:30, our model now says that we will have to wait 20.33057851 minutes.

This table shows what the linear regression model will create: