6.100B: Recitation 5

Regressions, Overfitting, & Machine Learning

May 5th, 2023

Actionables

PS3 Checkoff due at **5pm** today

PS4 Checkoff start today, due on **Friday, 5/12** at **5pm**

Microquiz 4 on **Wednesday, 5/10** in-class

Agenda

Curve Fitting
Linear Regressions
R^2
Overfitting

Machine Learning
Training / Testing Datasets
Evaluation

Let's get started!

https://mattfeng.tech/teaching/6.100B

Curve Fitting | Linear Regressions

The purpose behind curve-fitting is to take **data we have** and generalize it to a hypothesized **trend**

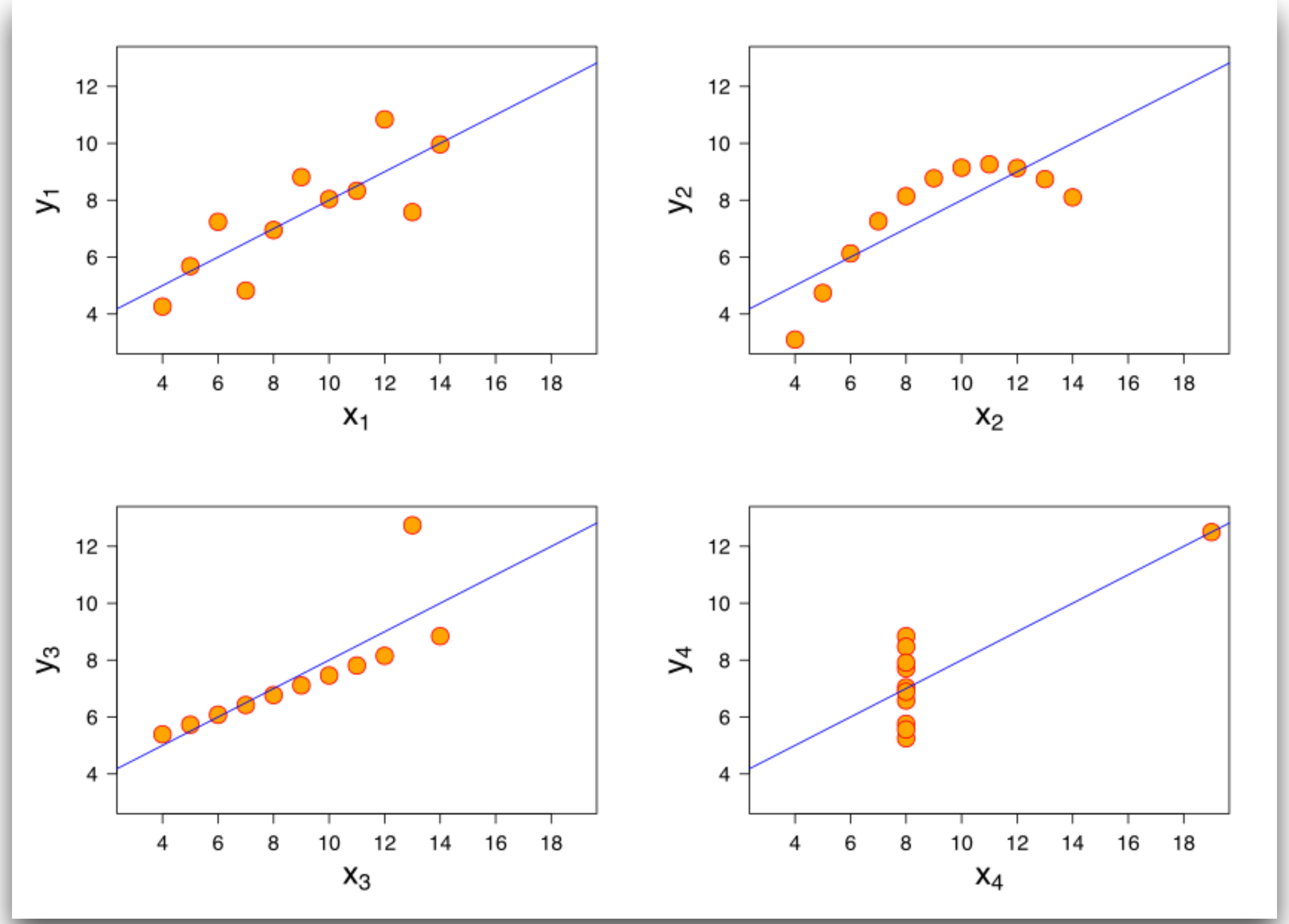
We can then use that **trend** to predict the nature of a system

It is an important way of mathematically creating and optimizing **hypotheses** based on data we're collecting

Curve Fitting | Linear Regressions

Anscombe's Quartet is an example of the significance of graphical analysis in producing generalizable models for data

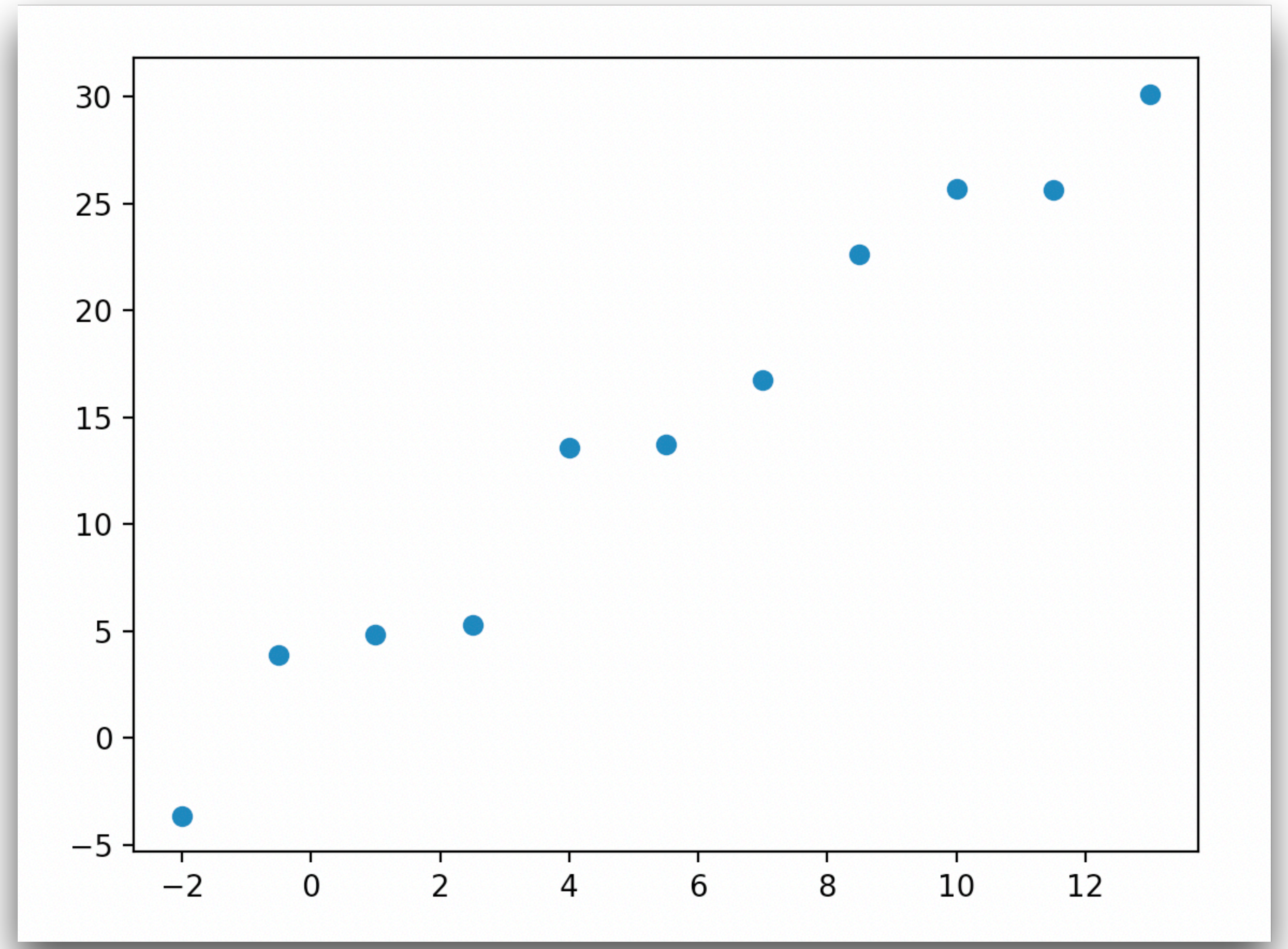
Each graph depicts a linear regression of the data in orange that minimizes our mean squared error (MSE)



Curve Fitting | Linear Regressions

How can we generate **general models** for fitting data?

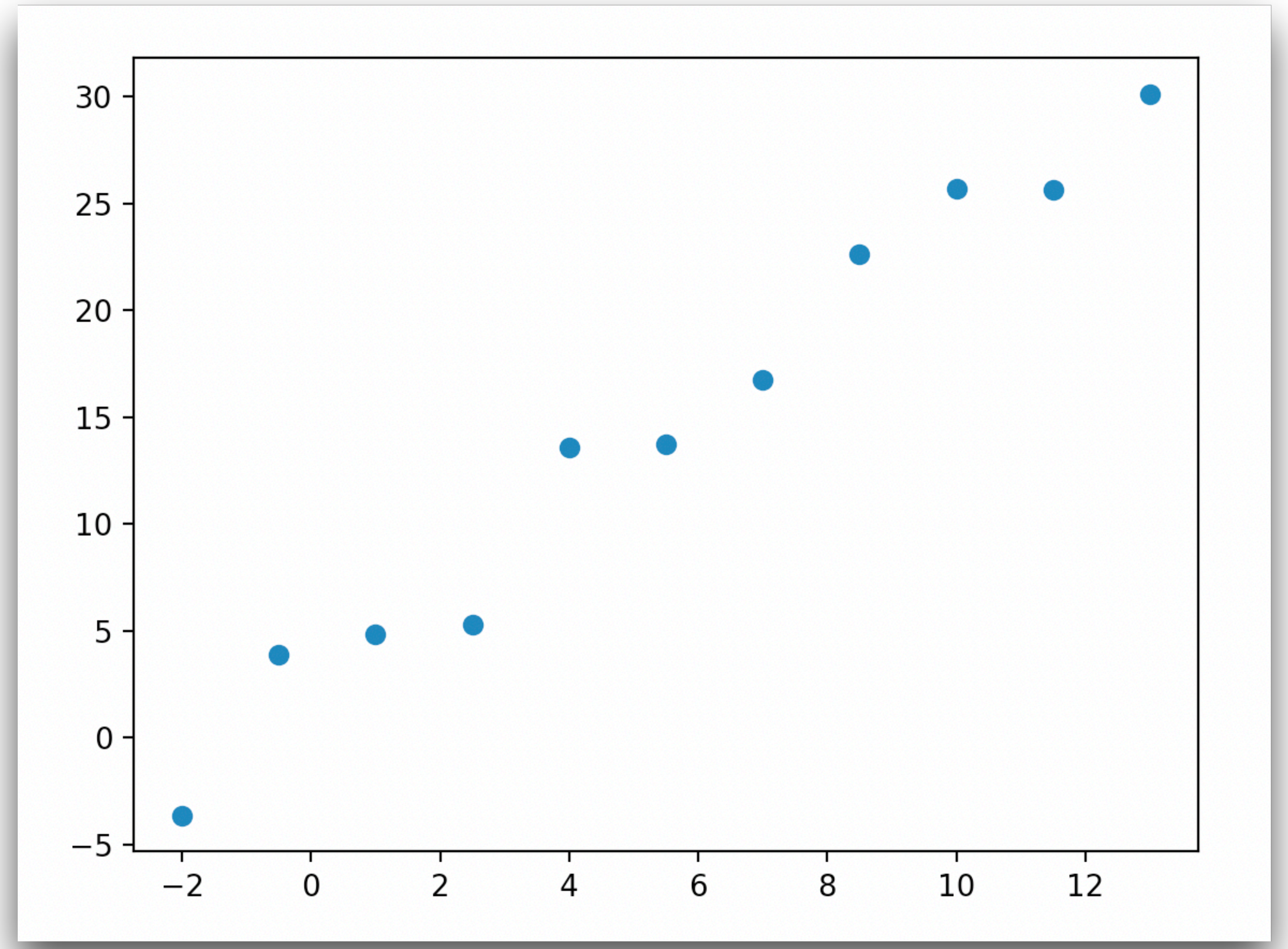
We'll think about this problem by asking **how do we minimize the error between our predictions and reality?**



Curve Fitting | Linear Regressions

Given a linear regression with a **single input variable** and a **single output variable**, there exists a curve that minimizes **the error**:

$$f(x) = \sum_{i}^{\infty} c_i x^i$$



Or if we ignore terms $i > 1$ (linear fit):

$$f(x) = c_1 x + c_0$$

Curve Fitting | Linear Regressions

What exactly are we minimizing? How do we choose coefficients c_1 and c_0?

We are looking to minimize the **error**, or **loss function**

A more informative measurement of a model's fit is actually R^2, or the **Coefficient of Determination**

$$R^2 = 1 - \frac{\text{sum squared regression (SSR)}}{\text{total sum of squares (SST)}}$$

Curve Fitting | Linear Regressions

$$R^2 = 1 - \frac{\text{sum squared regression (SSR)}}{\text{total sum of squares (SST)}}$$

$$R^2 = 1 - \frac{\sum_i^n (y_i - \hat{y}_i)^2}{\sum_i^n (y_i - \bar{y})^2}$$

Where $\hat{y}_i$ is our **predicted value** for input i, and $\bar{y}$ is our **average output**

Curve Fitting | Linear Regressions

We can see that maximizing R^2 means **minimizing the SSR**

SSR is the **mean squared error** of our sample batch:

$$SSR = \sum_{i}^{n} (y_i - \hat{y}_i)^2$$

$$SSR = \sum_{i}^{n} (\text{observed}[i] - \text{predicted}[i])^2$$

Curve Fitting | Linear Regressions

We can define our **objective function** (also called a **loss function**) we'd like to **minimize** as our SSR divided by the number of samples to effectively minimize our **per sample error**

$$J(c) = \frac{1}{N}\sum_{i}^{N}\left(y_i - \hat{y}_i\right)^2$$

Or for our linear model:

$$J(c_1, c_0) = \frac{1}{N}\sum_{i}^{N}\left(y_i - (c_1 x_i + c_0)\right)^2$$

Curve Fitting | Linear Regressions

One way to minimize our objective function $J(c_1, c_0)$ is with **gradient descent**.

We can initialize our coefficients with random values, and adjust them based on **how much they contribute to the error**

To update these coefficients, we can subtract their contribution to the error:

$$c_i := c_i - \epsilon \frac{\partial J(c)}{\partial c_i}$$

Curve Fitting | Linear Regressions

$$c_i := c_i - \epsilon \frac{\partial J(c)}{\partial c_i}$$

Where ϵ is the **step size** in the direction of error. For stable adjustments, we want this to be small. For a linear model:

$$J(c_1, c_0) = \frac{1}{N} \sum_i^N \left(y_i - (c_1 x_i + c_0)\right)^2$$

$$\frac{\partial J(c)}{\partial c_i} = \frac{1}{N} \frac{d}{dc_i} \sum_j^N \left(y_j - c_1 x_j - c_0\right)^2$$

Curve Fitting | Linear Regressions

$$J(c_1, c_0) = \frac{1}{N}\sum_{i}^{N}\left(y_i - (c_1 x_i + c_0)\right)^2$$

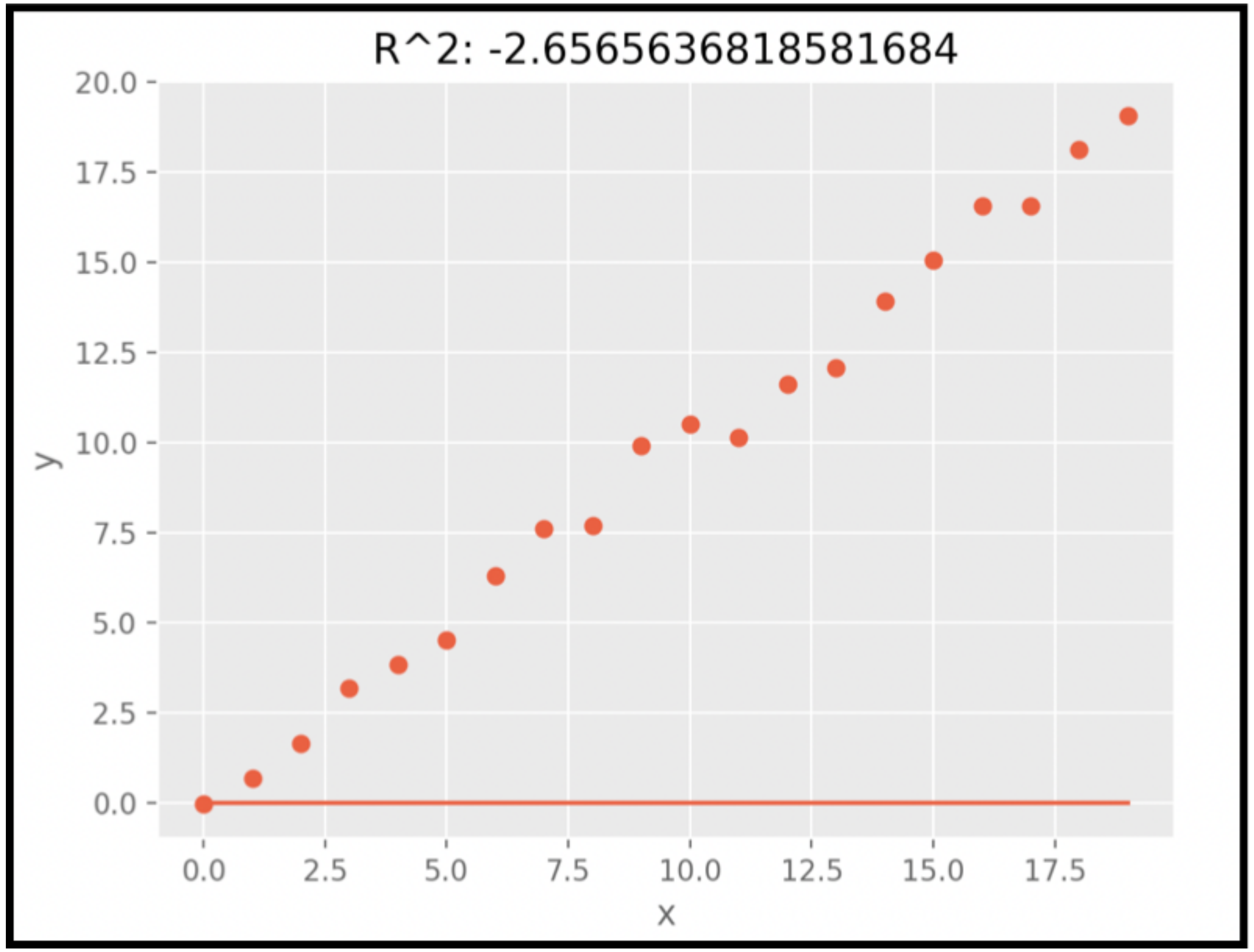
$$\frac{\partial J(c)}{\partial c_n} = \frac{1}{N}\frac{d}{dc_n}\sum_{i}^{N}\left(y_i - c_1 x_i - c_0\right)^2$$

$$c_1 := c_1 - \epsilon\left[-\frac{2}{N}\sum_{i}^{N}\left(y_i - c_1 x_i - c_0\right)x_i\right]$$

$$c_0 := c_0 - \epsilon\left[-\frac{2}{N}\sum_{j}^{N}\left(y_j - c_1 x_j - c_0\right)\right]$$

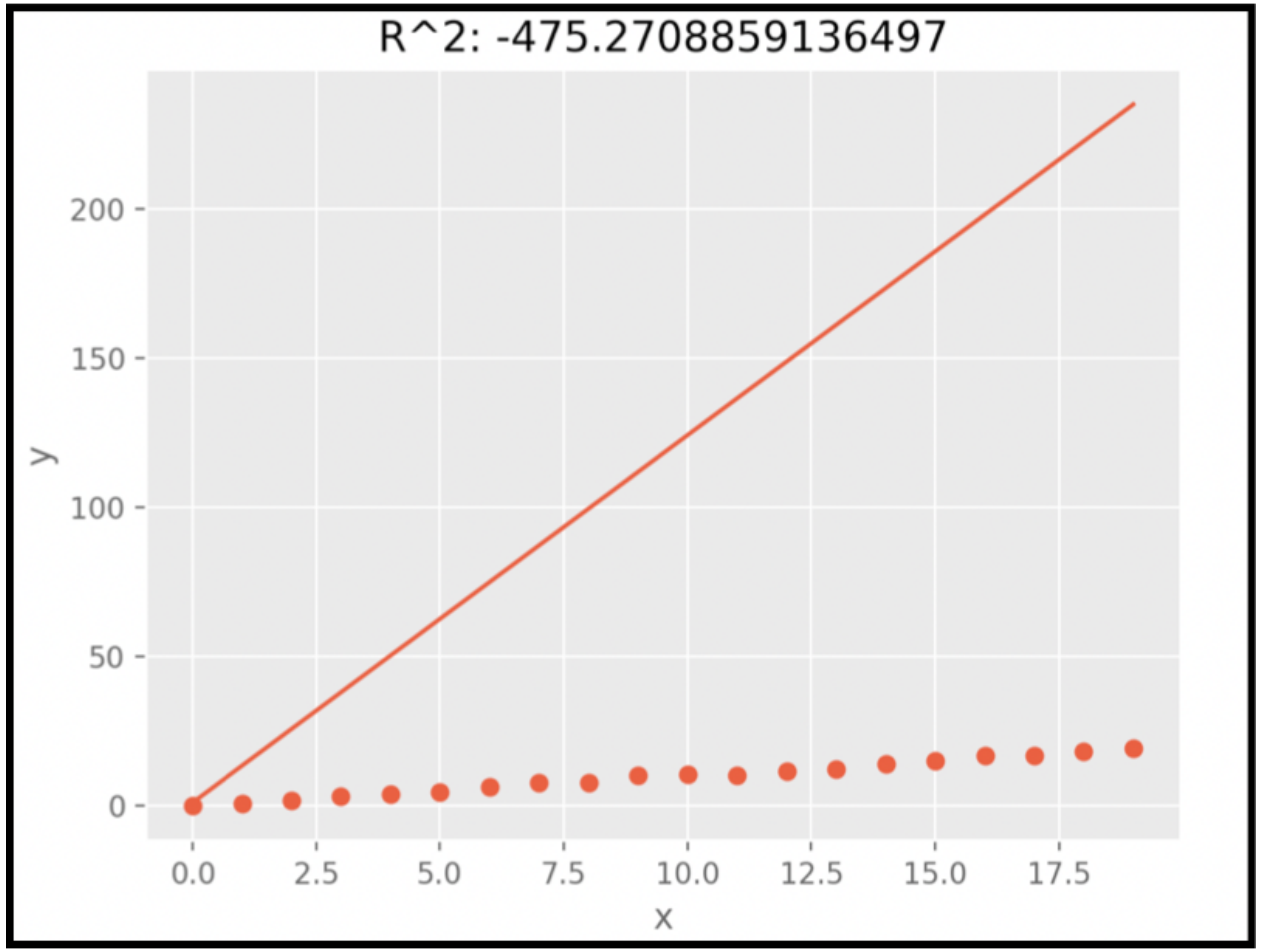
Curve Fitting | Linear Regressions

Let's try a **linear regression** on an Order-1 dataset with step-size 0.1:



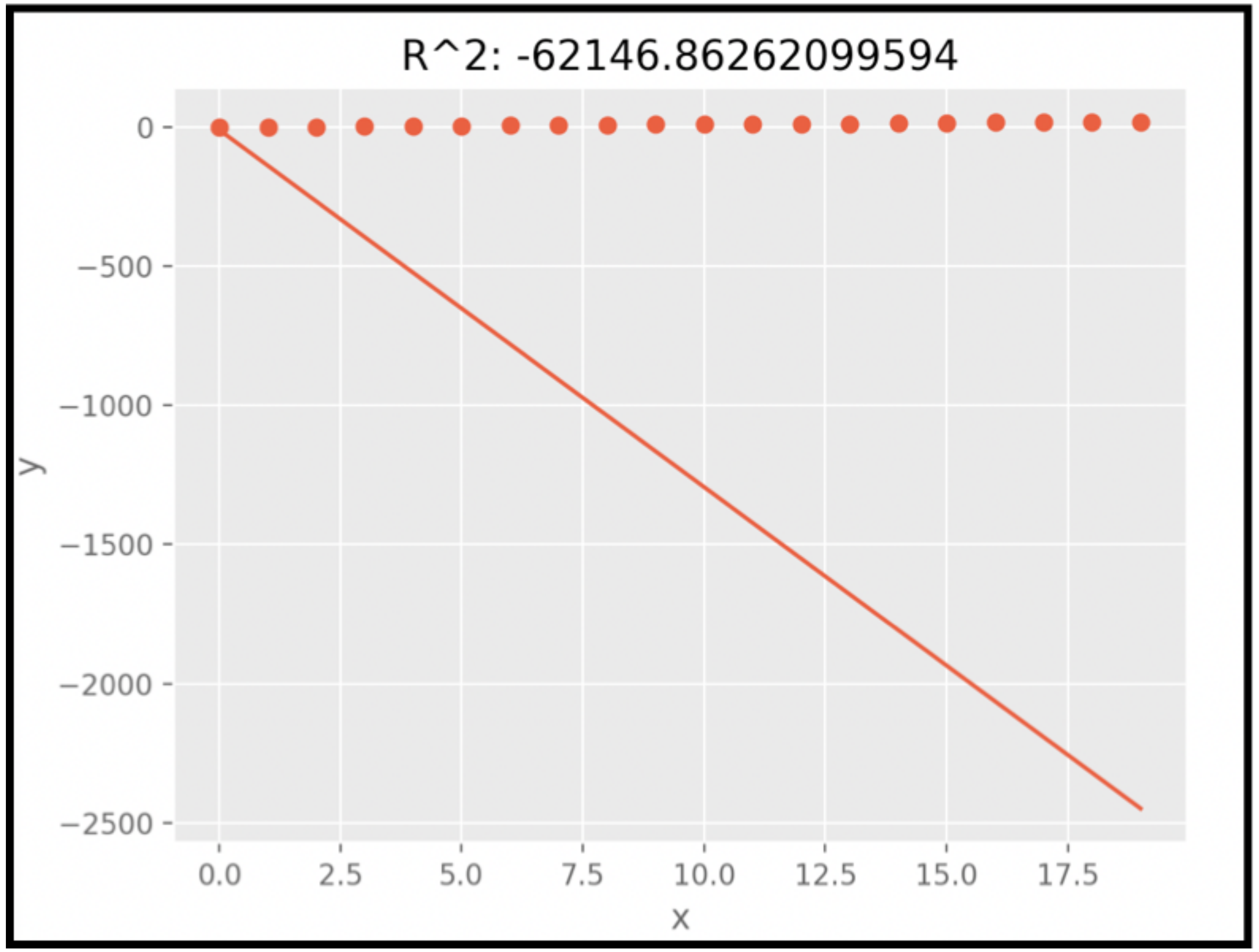
Curve Fitting | Linear Regressions

Let's try a **linear regression** on an Order-1 dataset with step-size 0.1:



Curve Fitting | Linear Regressions

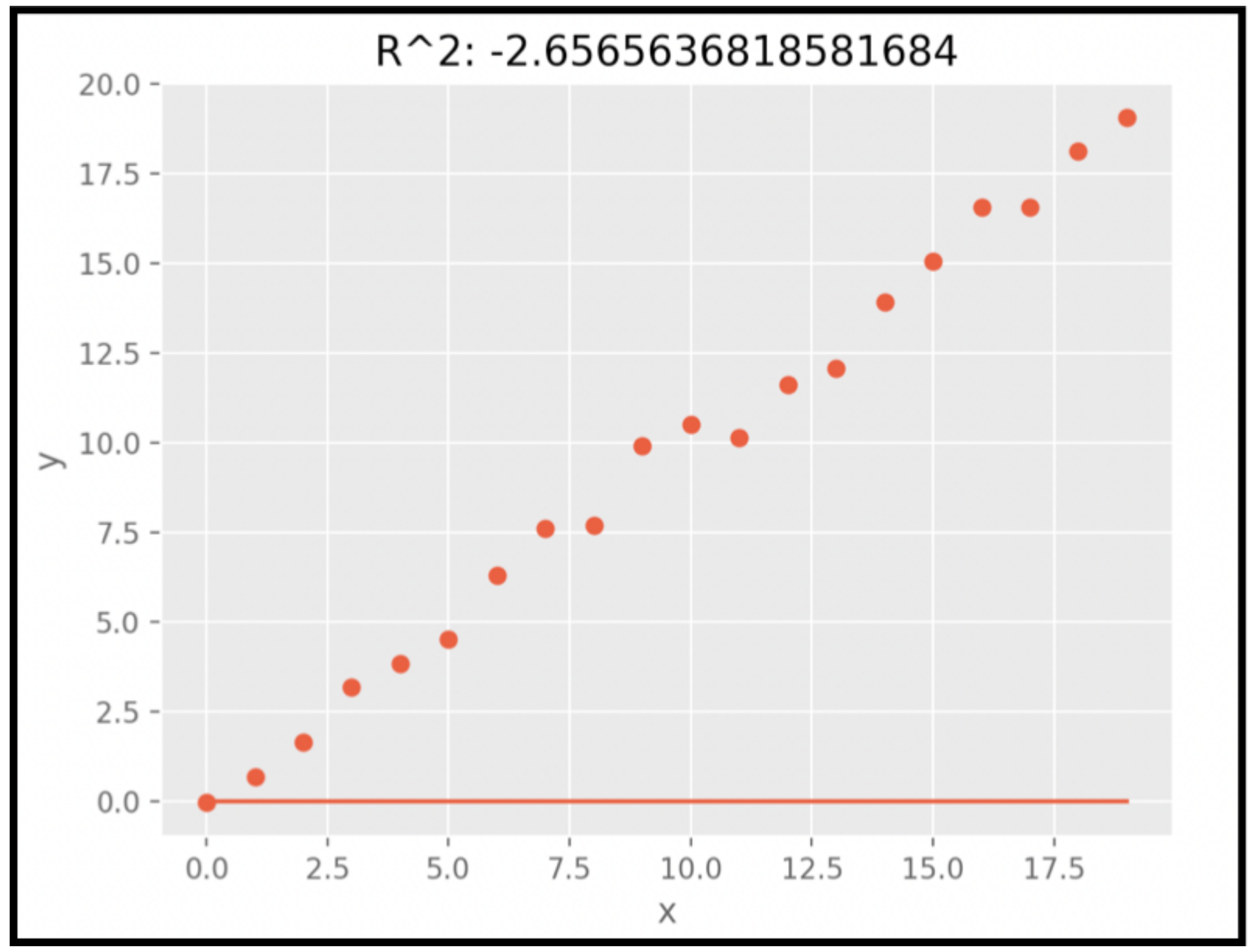
Let's try a **linear regression** on an Order-1 dataset with step-size 0.1:

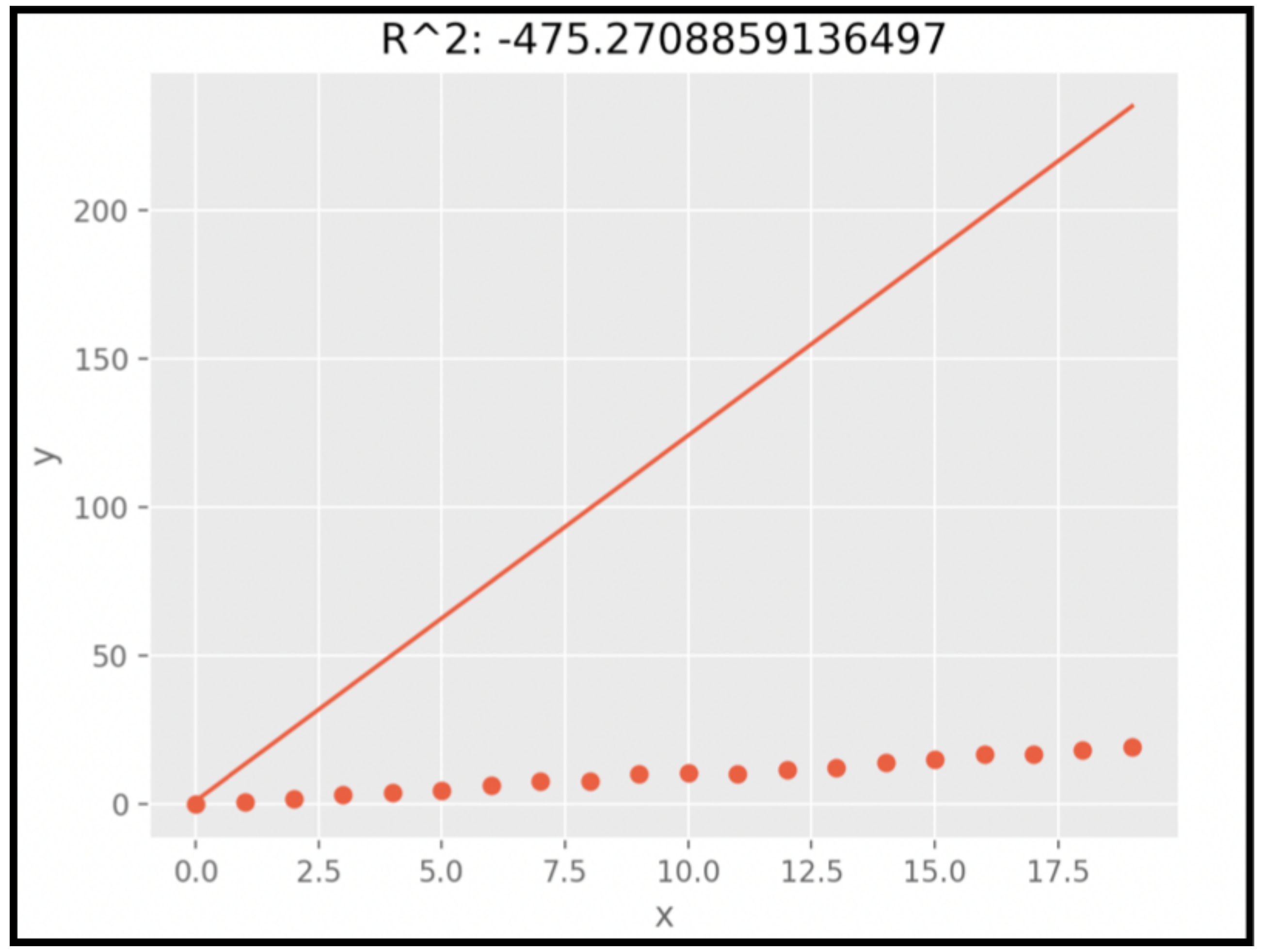


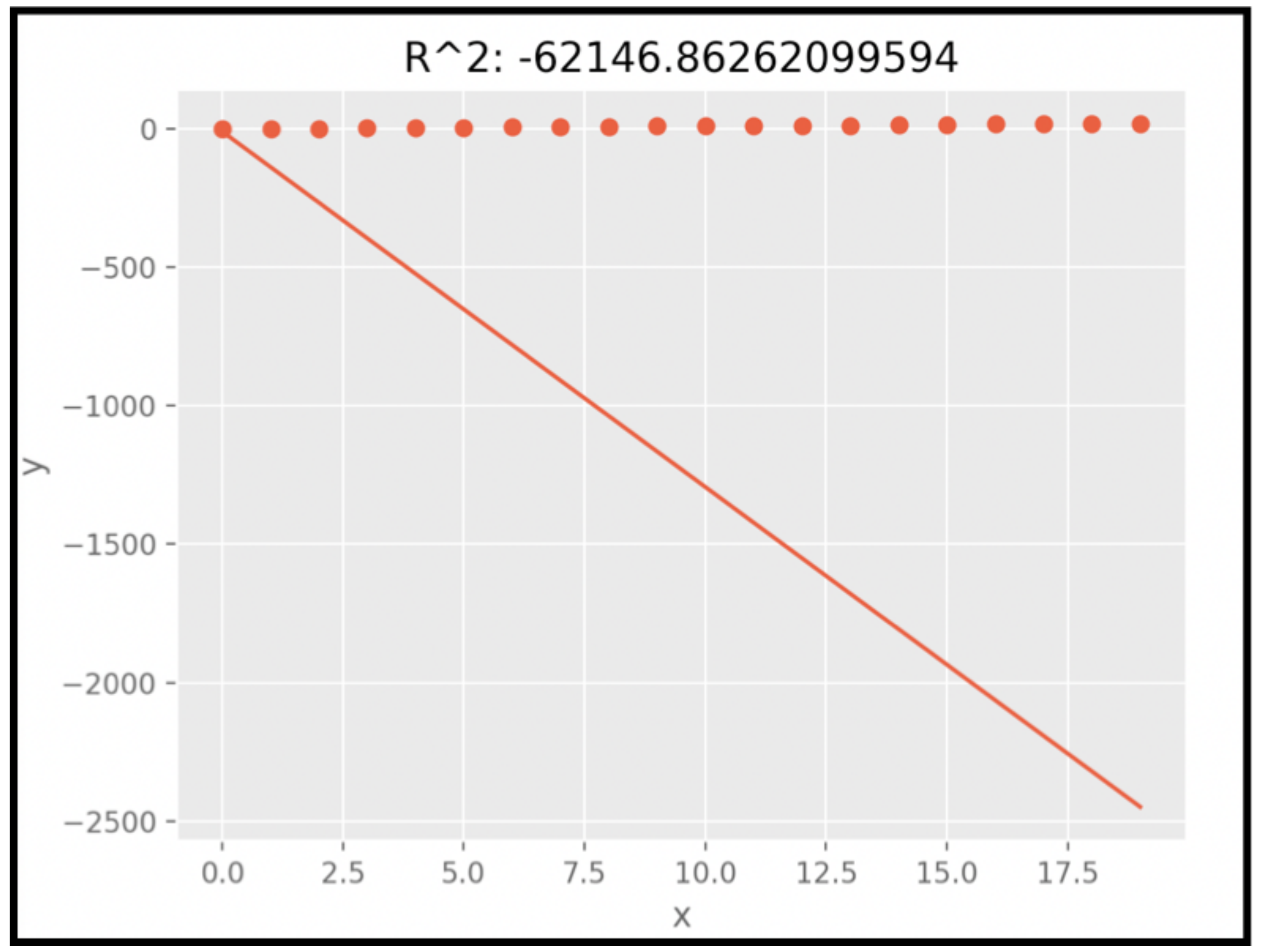
Curve Fitting | Linear Regressions

What's happening here?

Our **step size** is far too large, and our steps away from the the objective function overcompensate each coefficient's contribution!

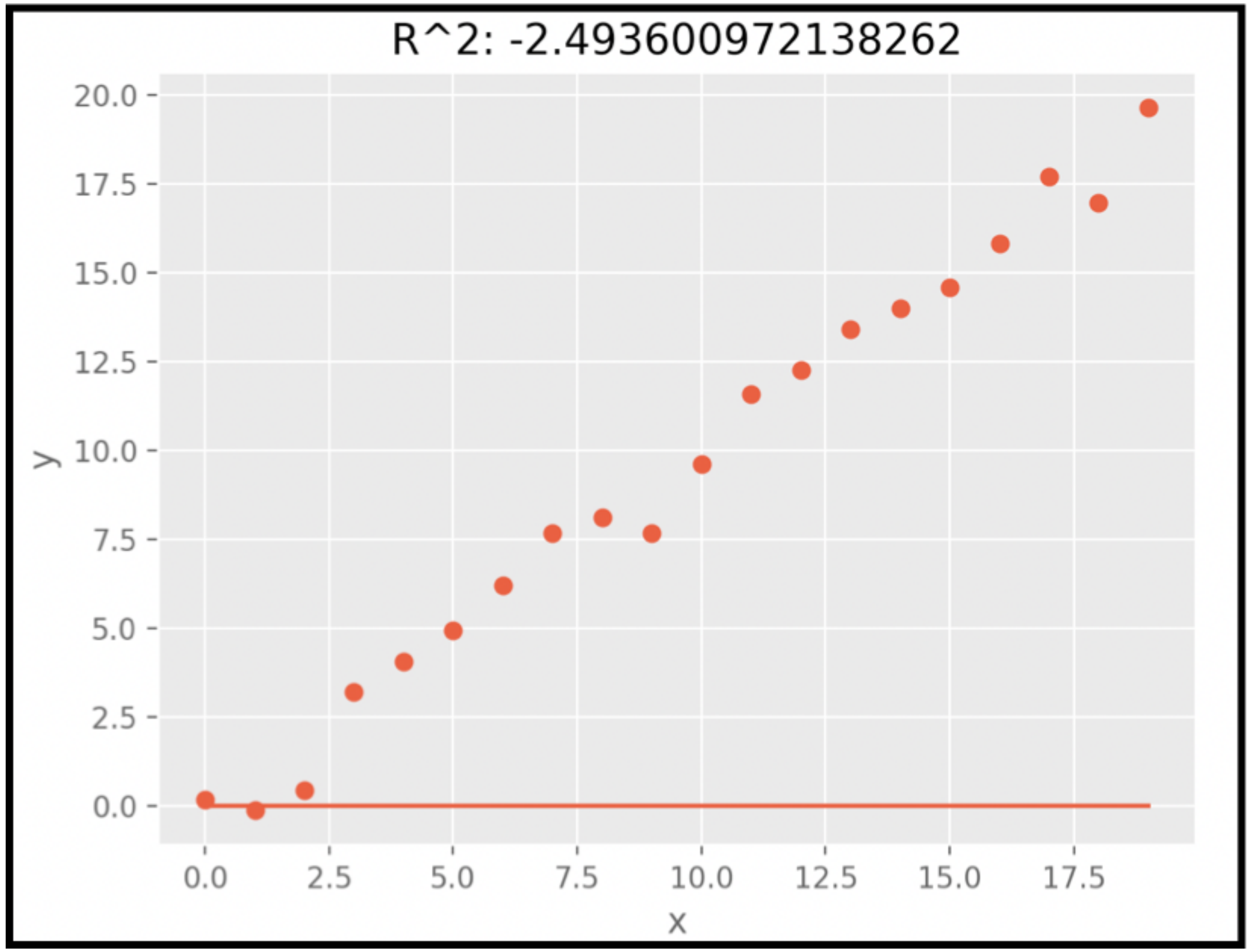






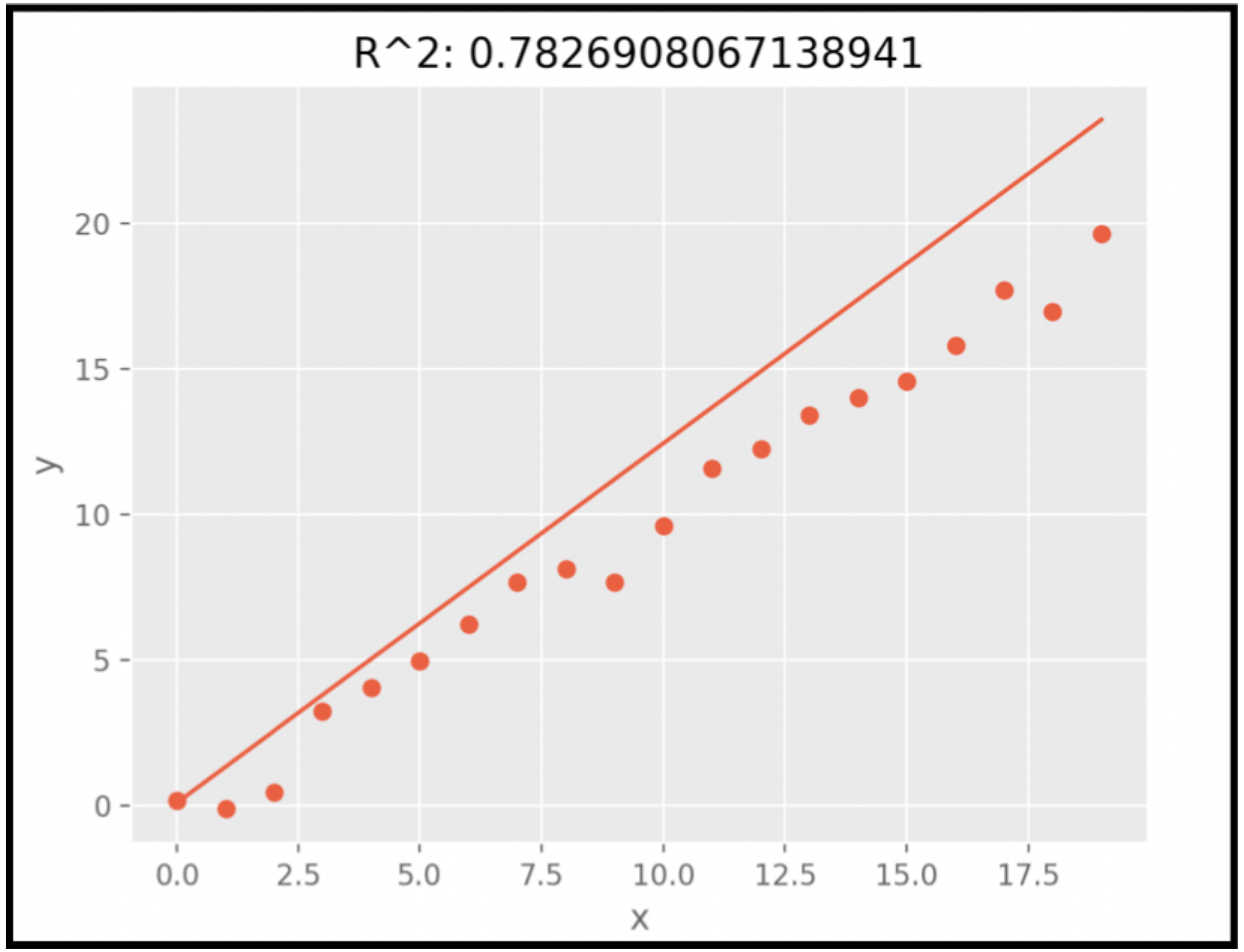
Curve Fitting | Linear Regressions

What about a step size of 0.01:



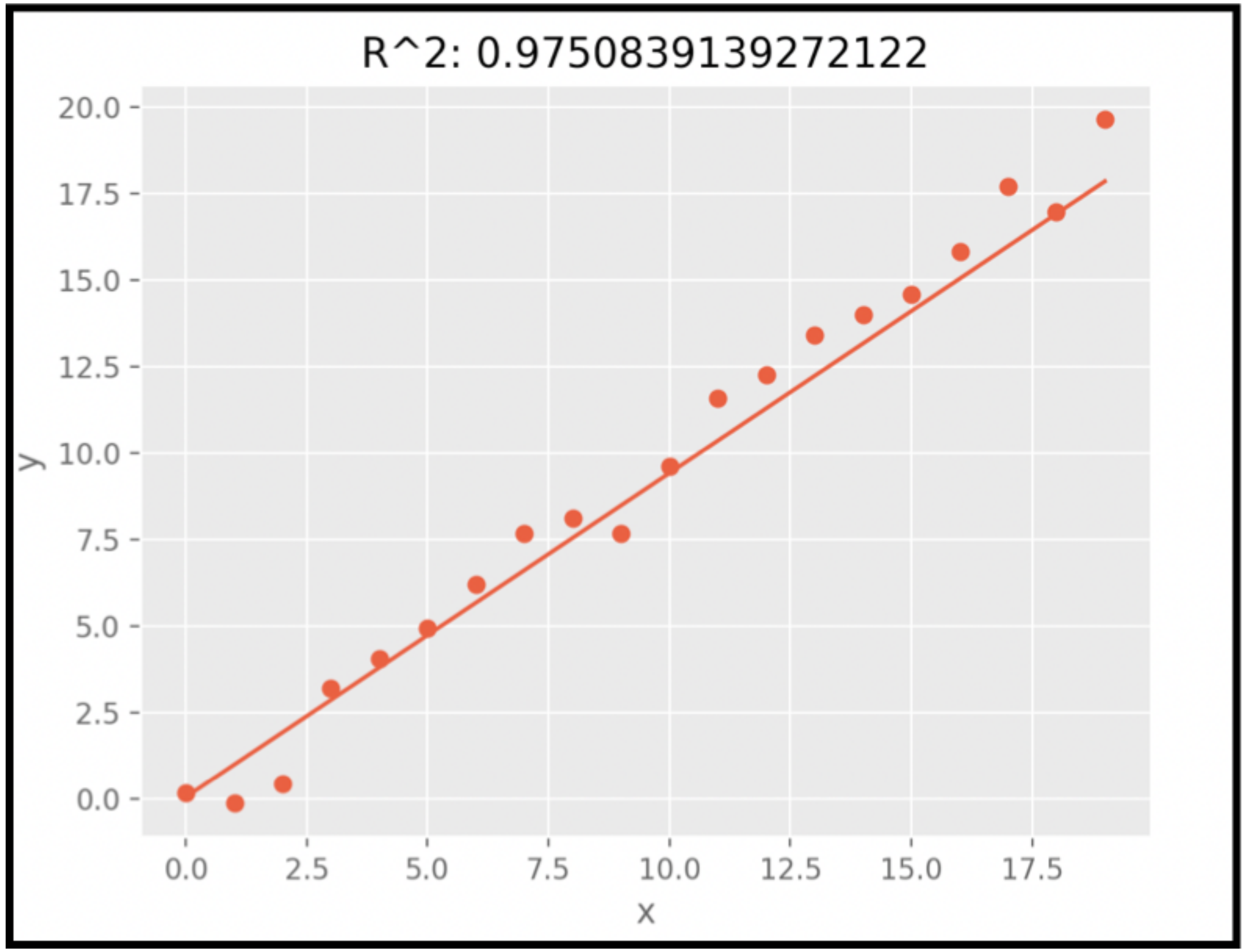
Curve Fitting | Linear Regressions

What about a step size of 0.01:



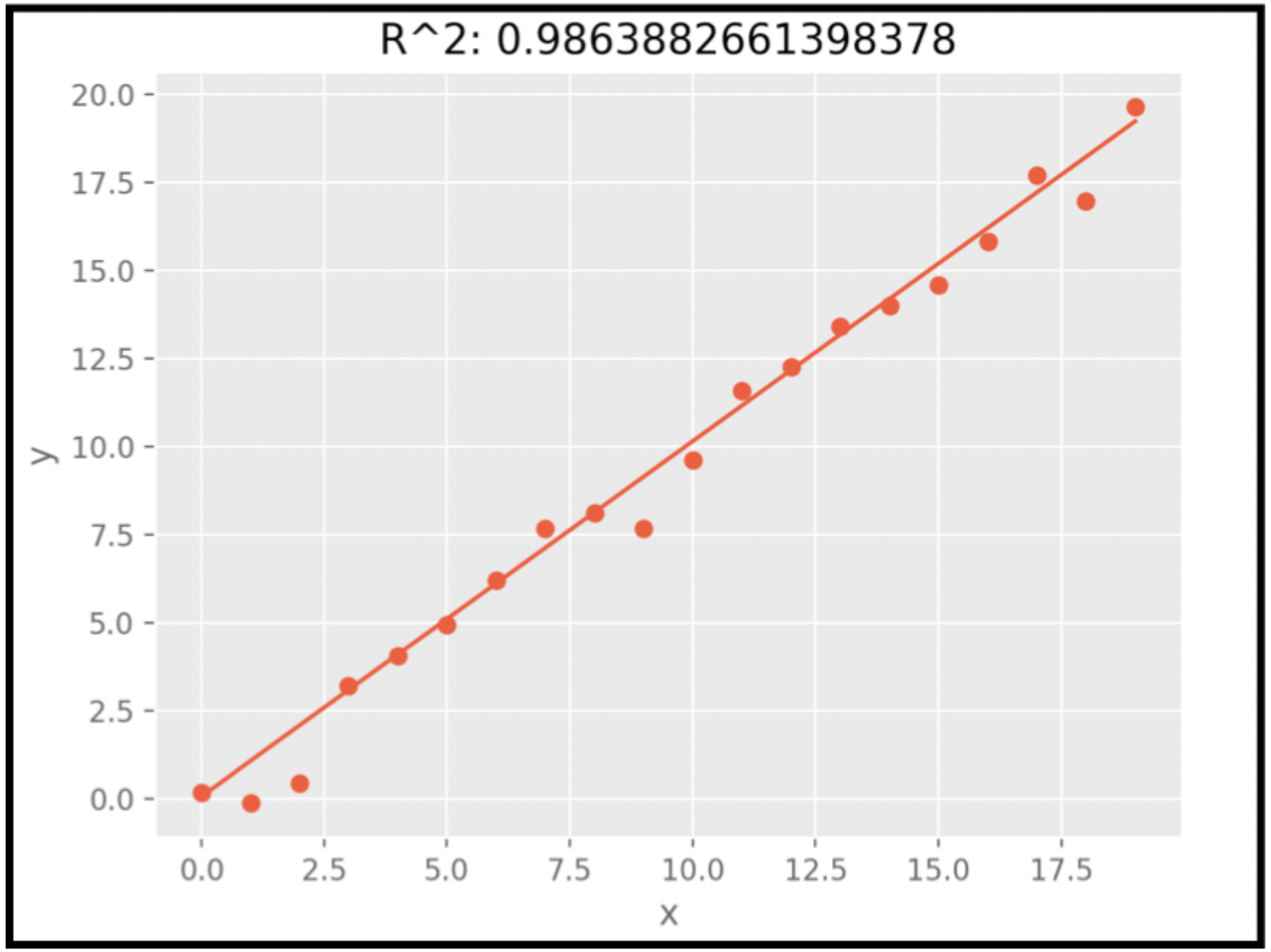
Curve Fitting | Linear Regressions

What about a step size of 0.01:



Curve Fitting | Linear Regressions

After only **3 passes** of our data, our model has a great fit!



Curve Fitting | Linear Regressions

We can generalize this approach to systems of **higher degrees** as well!

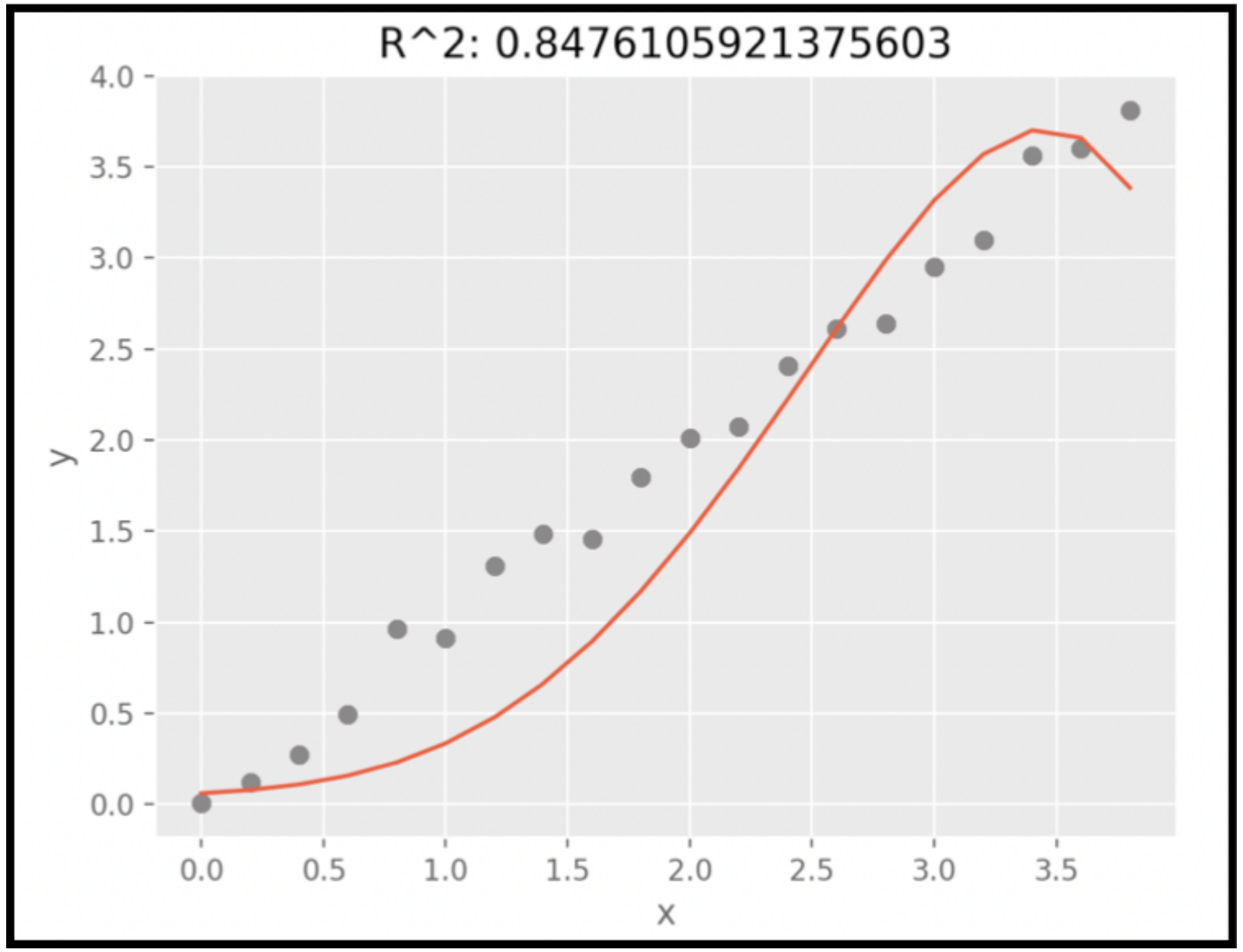
$$J(c_2, c_1, c_0) = \frac{1}{N}\sum_{i}^{N}\left(y_i - (c_2 x_i^2 + c_1 x_i + c_0)\right)^2$$

We take the same approach for updating coefficients of higher degrees:

$$c_2 := c_2 - \epsilon\left[-\frac{2}{N}\sum_{i}^{N}\left(y_j - c_2 x_i^2 - c_1 x_i - c_0\right)x_i^2\right]$$

Curve Fitting | Linear Regressions

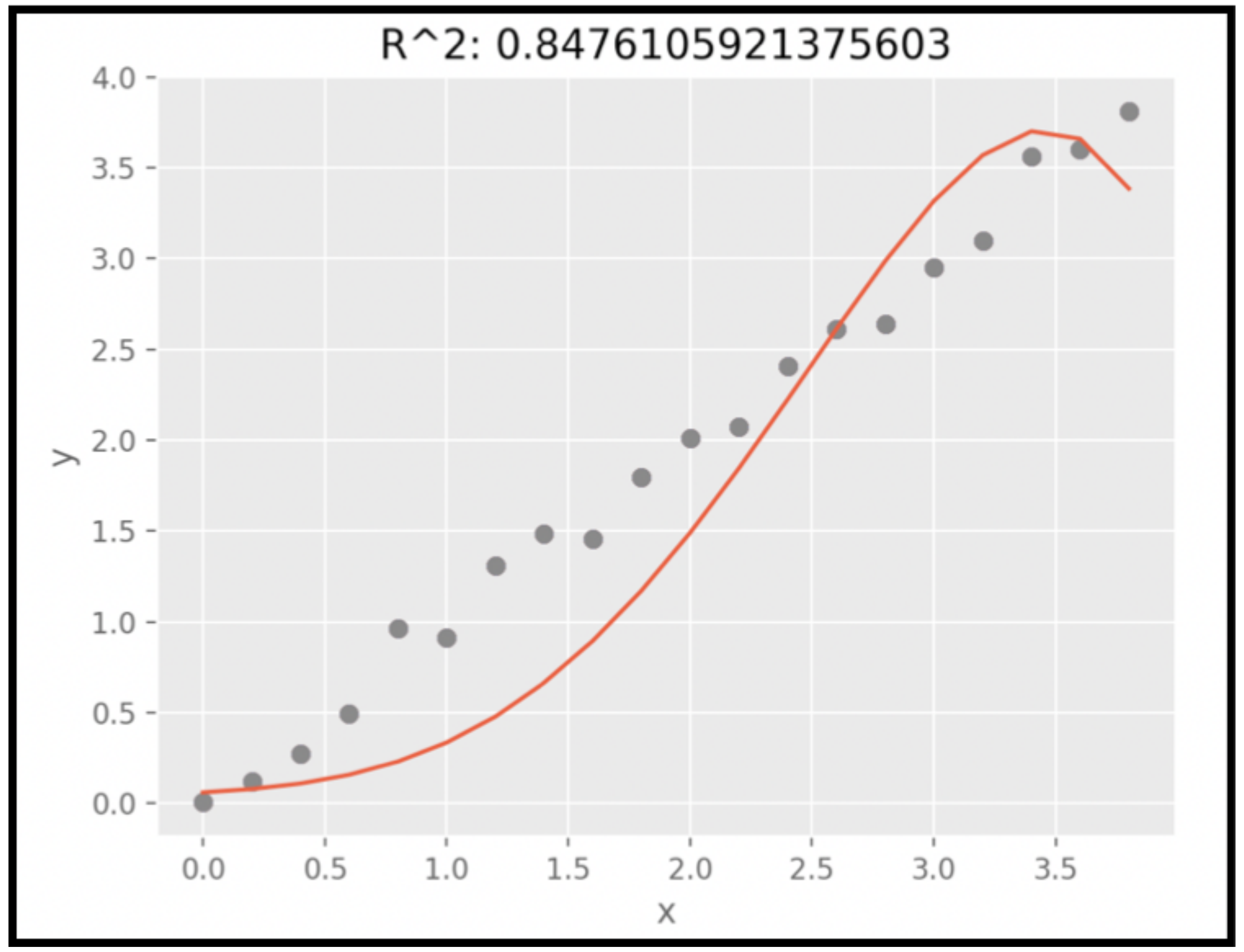
In this example we take a **5th degree** linear regression on a noisy **1st degree** dataset:



Curve Fitting | Linear Regressions

Although we have a high R^2, and potentially a lot higher than the linear fit if we keep training, the model is **not generalizable**

If we train and test on the same data, and our model is infinitely complex enough to directly touch each datapoint, we'll converge on $R^2 = 1$



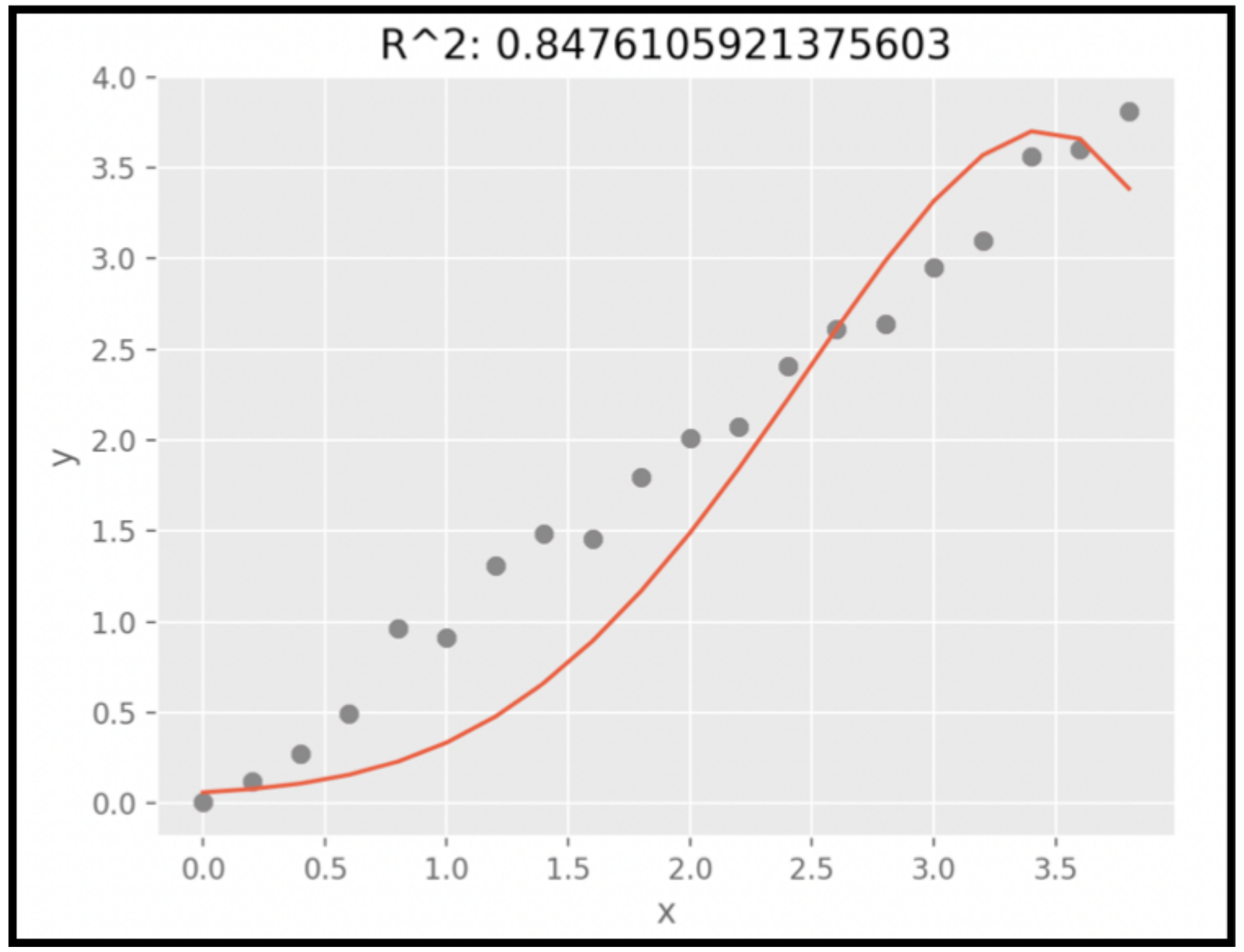
But we know this is bad because our data often contains **noise**!

Curve Fitting | Overfitting

This phenomenon is known as **overfitting**, and when we only train a model on the same **training data**, we're fitting it to that dataset, and not the system that it comes from

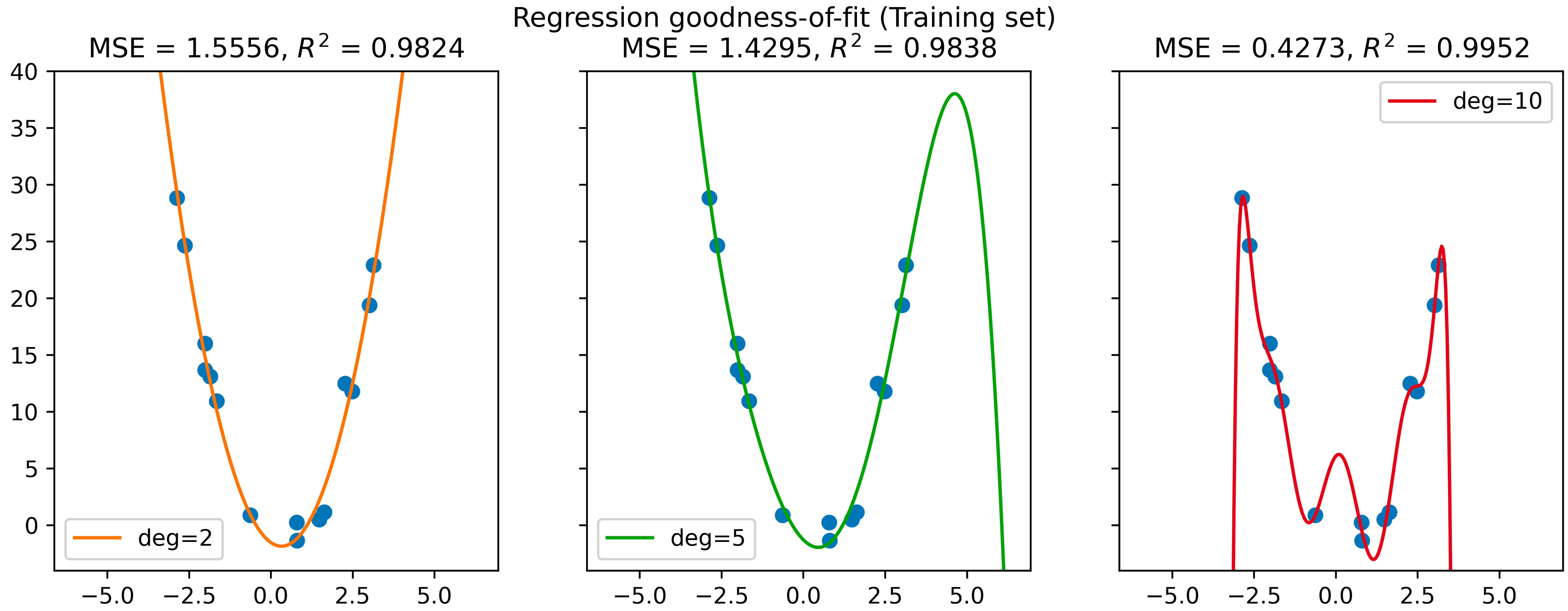
Therefore, we must evaluate the model on data we **did not train on**

We need to **dissuade our model from overfitting**

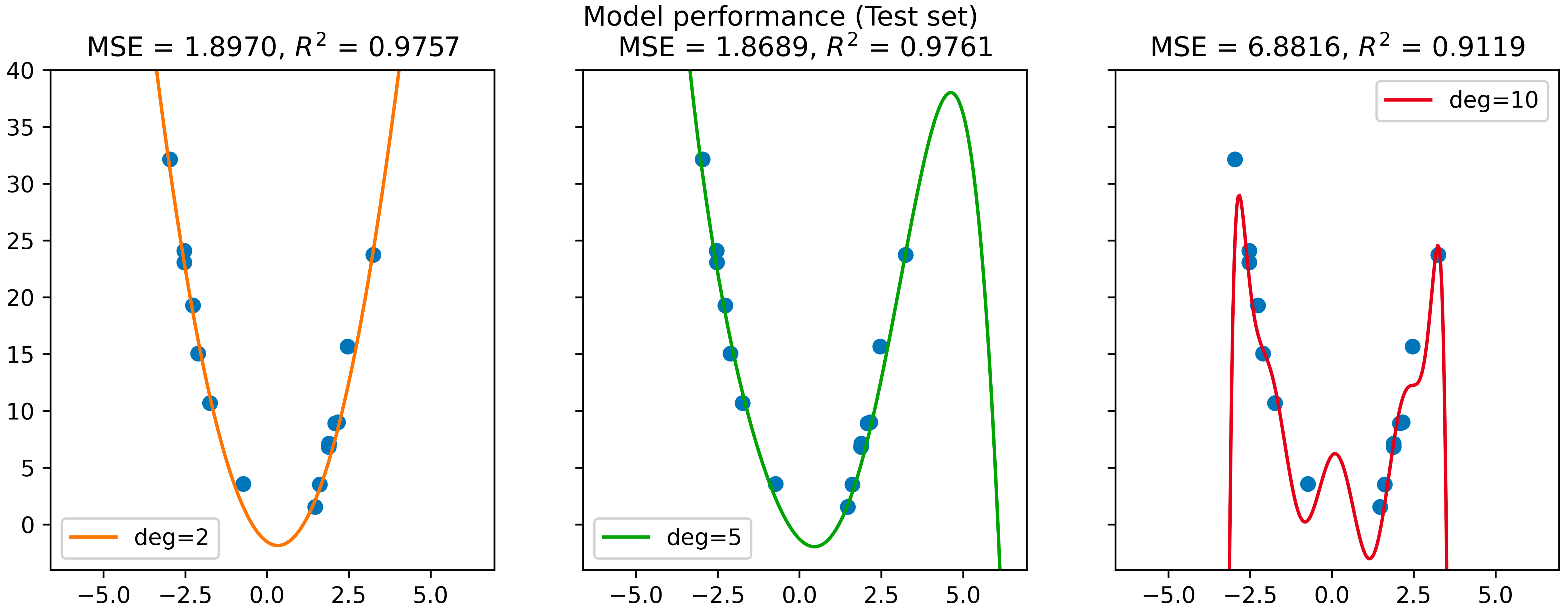


Curve Fitting | Overfitting

To prevent **overfitting** and keep our model generalizable, we have to tune the degree of our model to the shape of the data, and how much data we have (**hyperparameter tuning**)



Curve Fitting | Overfitting



Curve Fitting | Machine Learning

Although the 10th-order model had a better R^2 than the 2nd-Order on the training set, it performed much worse when exposed to new testing data

The ultimate goal of regression is to create a model that hypothesizes the behavior of a system (maybe even linearizes a particular range by using a lower order model)

The higher order data may end up using much of its “extra” dimensionality to fit the noise instead of the data, which will **not** generalize well as we introduce more data!