INFSCI 2595: Introduction to Machine Learning

Week 1: Introduction to regression

Fall 2024

Regression deals with continuous responses

- In simple terms, a continuous variable is a decimal number
- More formally, the responses are real-valued, meaning there are an infinite number of real numbers within any given interval.
- Your trained model will output a prediction that can be any floating-point number
- The goal of regression is to find a continuous function that maps every possible value of the input to a corresponding output.
- We'll talk a lot about "linear regression," but you'll see that we are not limited to learning a simple linear trend

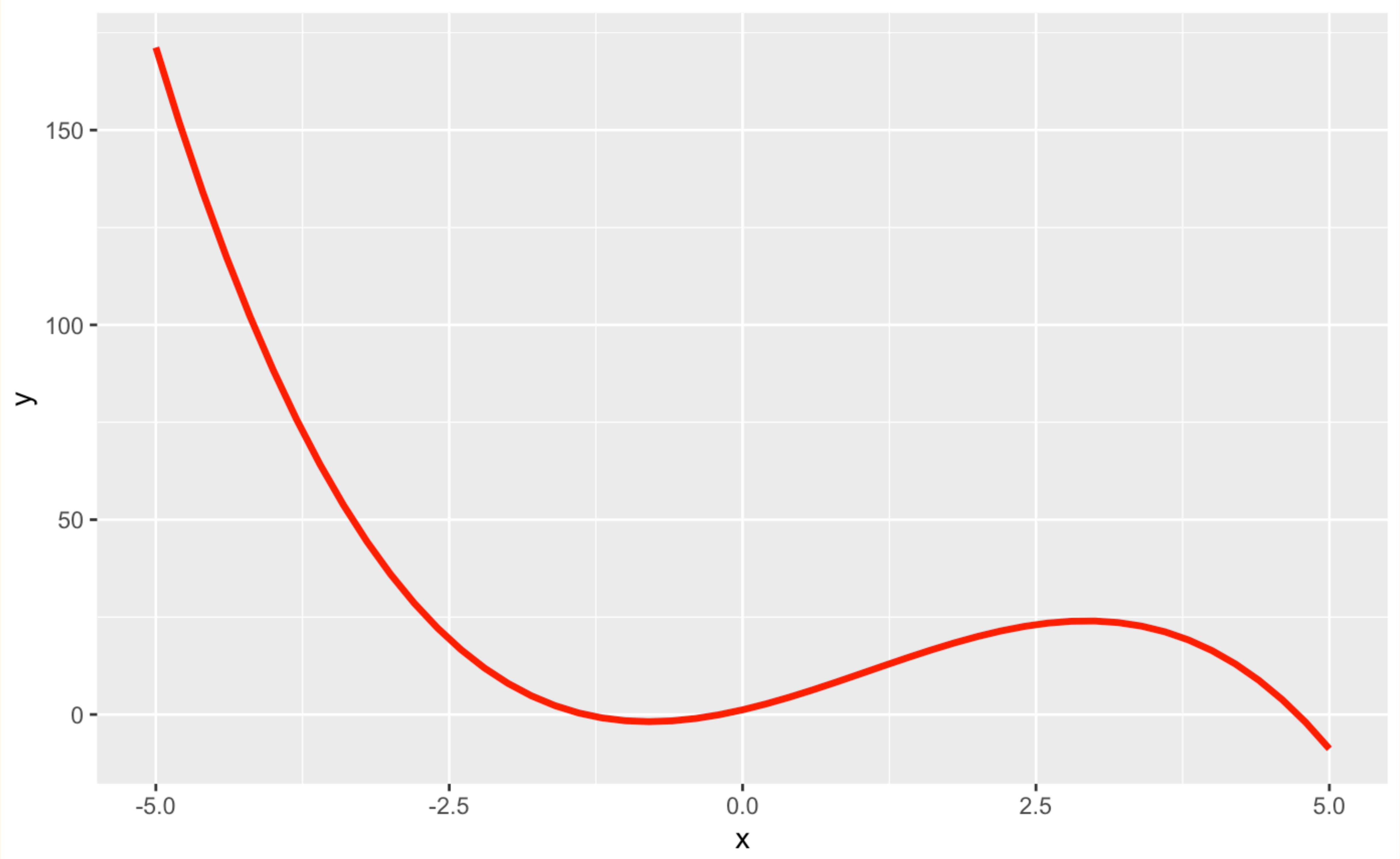
Let's work through a concrete, but heavily simplified example

- **Recall that all relationships learned from data are approximations:** $y \approx f(\mathbf{x})$
- **However for this example only, let's create data where we know the ground truth - we will define the noise-free relationship between the single input variable and the response**
- **The true relationship will be a cubic function with the following 4 coefficients:**

$$y = 1.2 + 7x + 3.2x^2 - x^3$$

$$\beta_{TRUE} = \left[1.2,\ 7,\ 3.2,\ -1\right]$$

The true relationship is cubic



- In reality, we never see the **true, noise-free signal.** It is hidden from us
- We see a corrupted signal, where datapoints differ from the ideal line due to chance and errors in measurement

- Remember this is **FAKE** data: no real-life process can be defined exactly, without error, by a mathematical expression

For this example, let's generate synthetic noisy samples based on the true trend

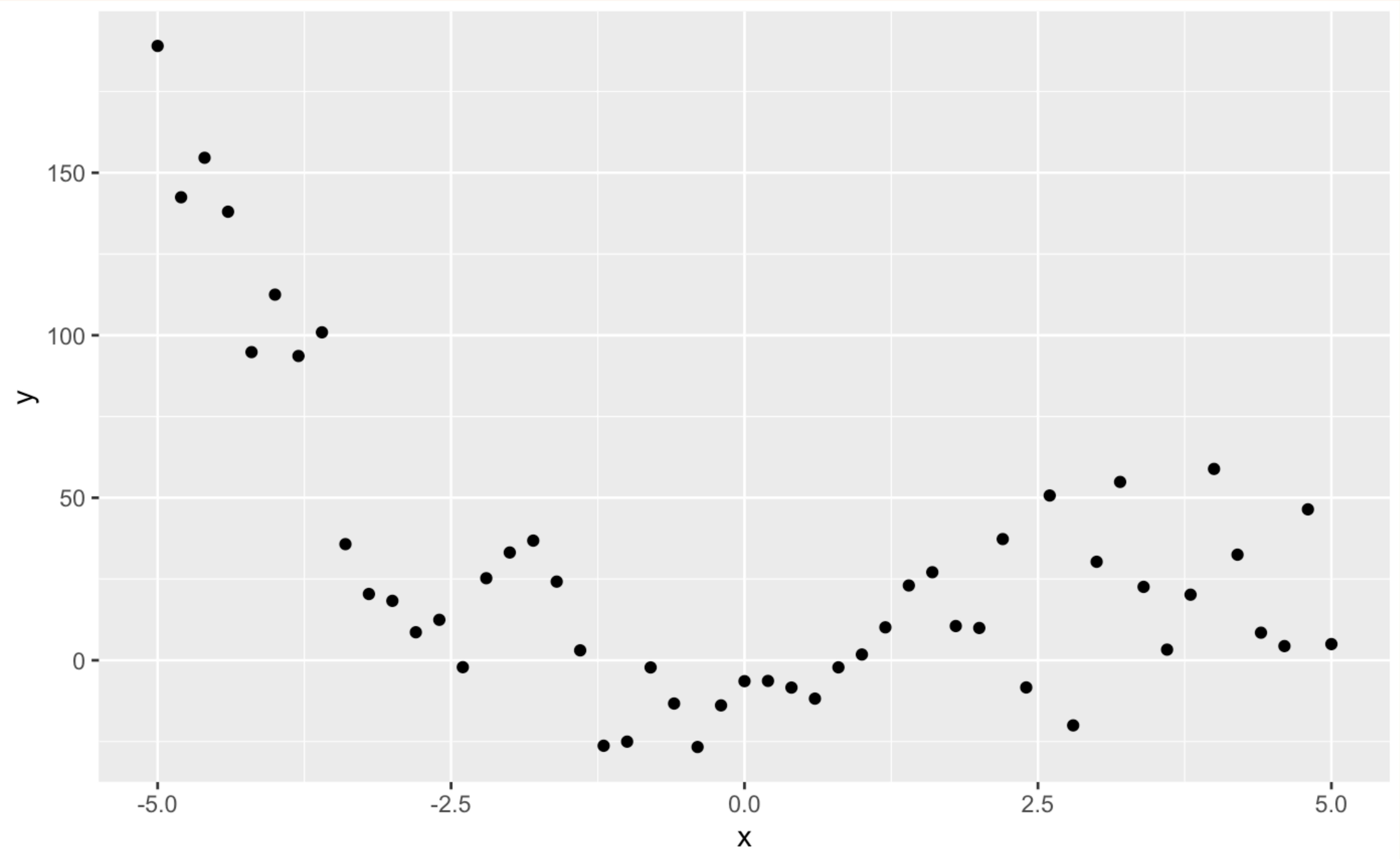
- Let us imagine that we have collected $N = 51$ noisy observations
  - Denote the observed inputs as: $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$
  - Denote the observed responses as: $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$
- Thus we collect $N = 51$ input-output pairs:

$$\{x_n, y_n\} \,\big|_{n=1}^{N=51}$$

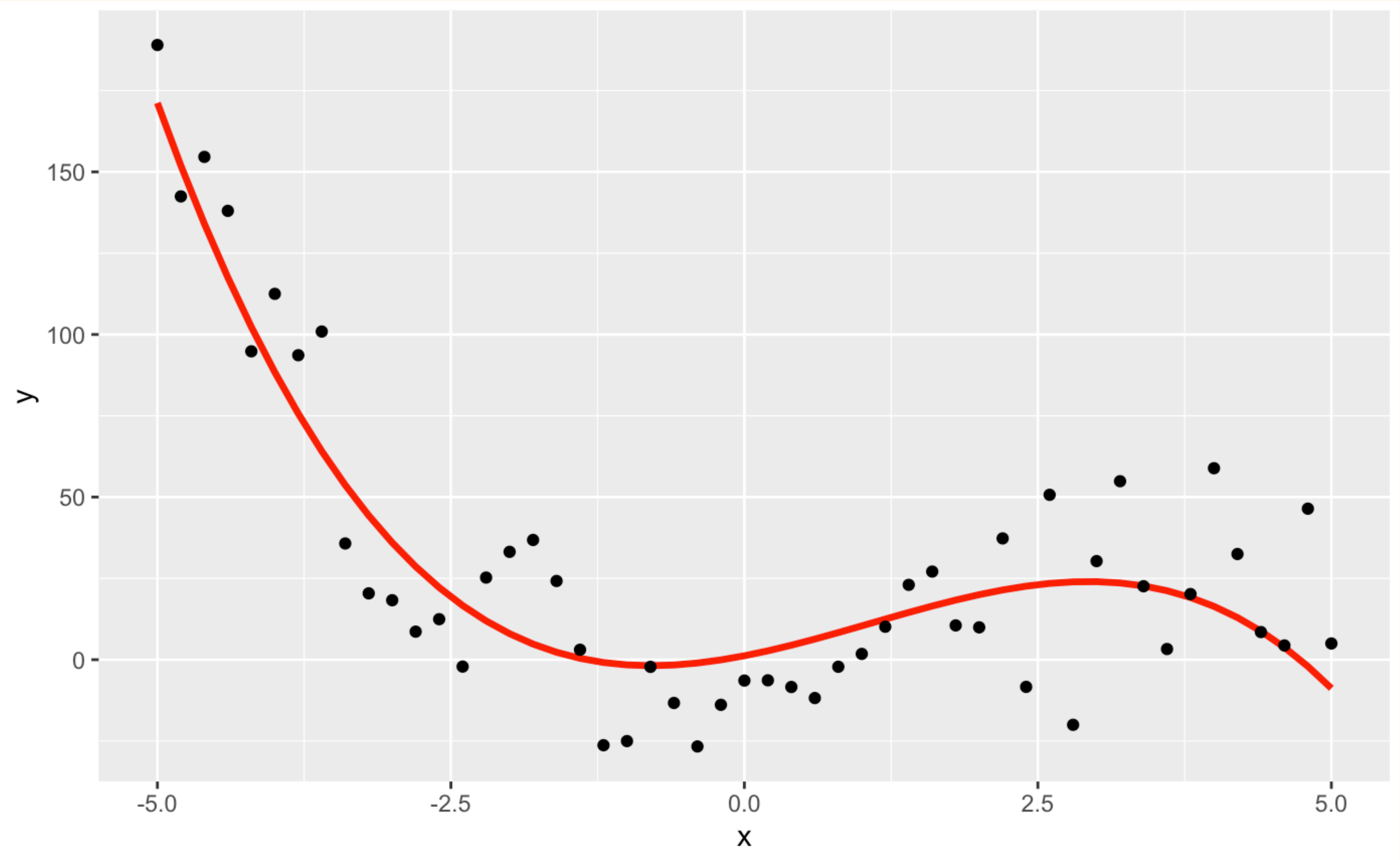
Since this is a toy problem, all the observations have been created using random number generators

- During the next portion of the course, we will discuss how these points can be generated from the parameterized function
- For now, just imagine that the 51 input-output pairs have been collected independently

If the true trend were not shown, can you immediately tell the “true relationship”?



Scatter plot showing the 51 noisy observations as black markers. Truth is still displayed as the red parabola



Now let's assume we do not know the truth, and all we have are the inputs and their corresponding outputs

- We want to <u>train</u> or <u>fit</u> a model between the response y and the input x
- We will start out with a simple linear relationship (we don't know that it's cubic yet!)

$$y = \beta_0 + \beta_1 x + \textbf{error}$$

- Always learning approximations - you must expect and account for error
- The task of machine learning is to estimate the values of the parameters from the data by minimizing the observed error between the model's predictions and the true labels

We will discuss the exact details of the learning process later in the semester

- For now, let's make use of R's packages to handle the math and statistics
- Data is stored in an object called `df` which contains two columns (variables) named x and y
- Fit a linear relationship in R using the `lm()` function and the formula interface.

Print out a few rows of the df object to show the variables we are working with

```{r}
df |>
  dplyr::select(x,y) |>
  slice(1:20)
```

A tibble: 20 × 2

x <dbl>	y <dbl>
-5.0	171.200
-4.8	151.920
-4.6	134.048
-4.4	117.536
-4.2	102.336
-4.0	88.400
-3.8	75.680
-3.6	64.128
-3.4	53.696
-3.2	44.336

1–10 of 20 rows

- The RFDS book has an excellent introduction to dplyr and the pipe operator |> (or, %>%)
- Here we are using dplyr to select the columns named “x” and “y”, and then slice off the first 10 rows

Fit the linear model with `lm()`

The assignment operator <-
assigns an object to a variable

Set the data argument to `df`

```
lin_mod <- lm(formula = y ~ x, data = df)
```

Formula interface allows you to specify the response and inputs (more formally the predictors) to the model.

Basic expression: `<output variable names> ~ <input variable name>`

Read the expression as: “the output, y, is a function of the input, x”

Summarize the results of the model training with `summary()`

```r
lin_mod <- lm(formula = y ~ x, data = df)
lin_mod |> summary()
```

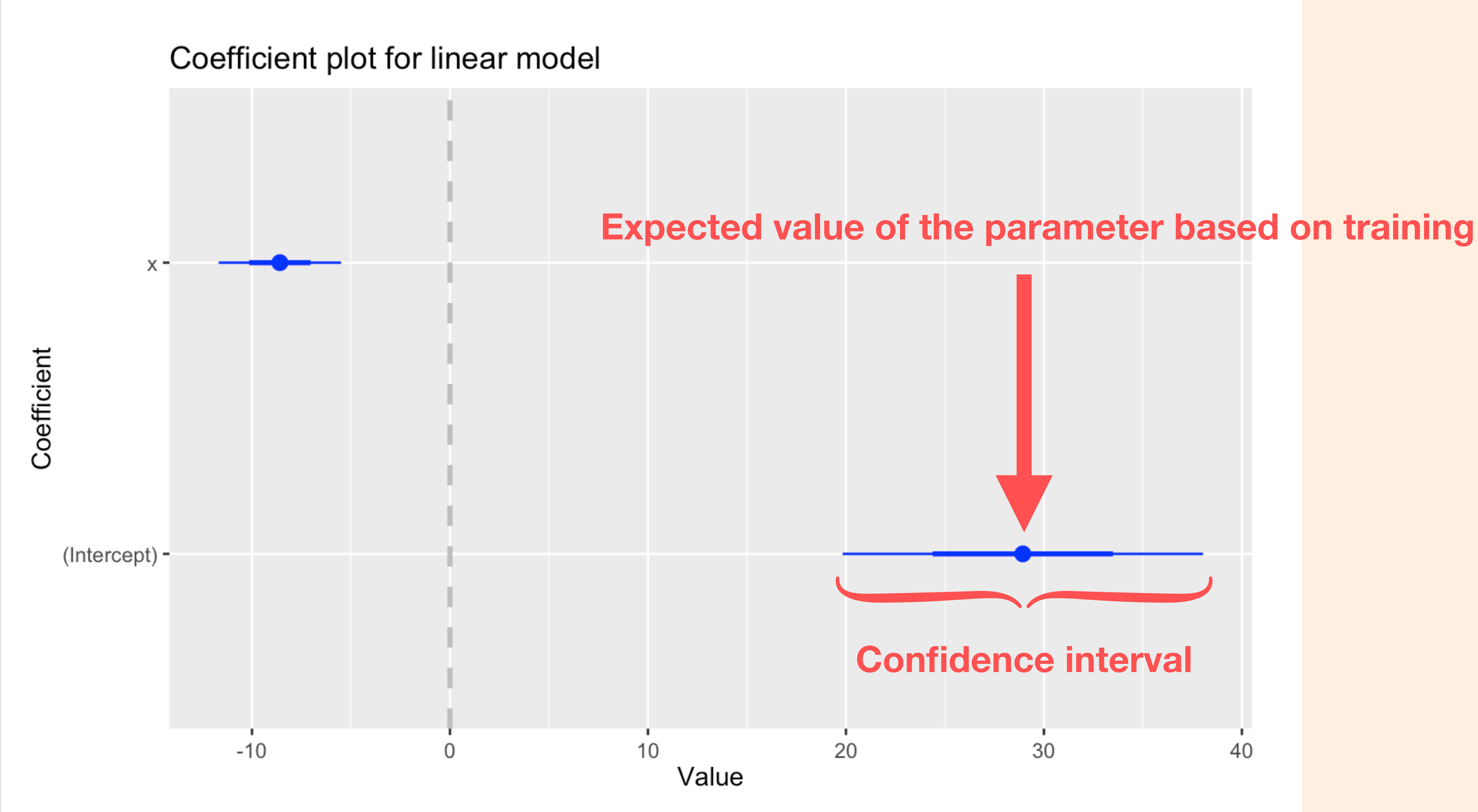
```
Call:
lm(formula = y ~ x, data = df)

Residuals:
   Min     1Q Median     3Q    Max 
-40.55 -28.40   1.31  19.59  99.31 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   28.933      4.550   6.358 6.55e-08 ***
x             -8.592      1.546  -5.559 1.11e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 32.5 on 49 degrees of freedom
Multiple R-squared:  0.3867,	Adjusted R-squared:  0.3742 
F-statistic:  30.9 on 1 and 49 DF,  p-value: 1.112e-06
```

β_0

β_1

Visualize the learned model parameters with `coefplot()`

```
coefplot::coefplot(lin_mod) +
  labs(title = 'Coefficient plot for linear model')
```



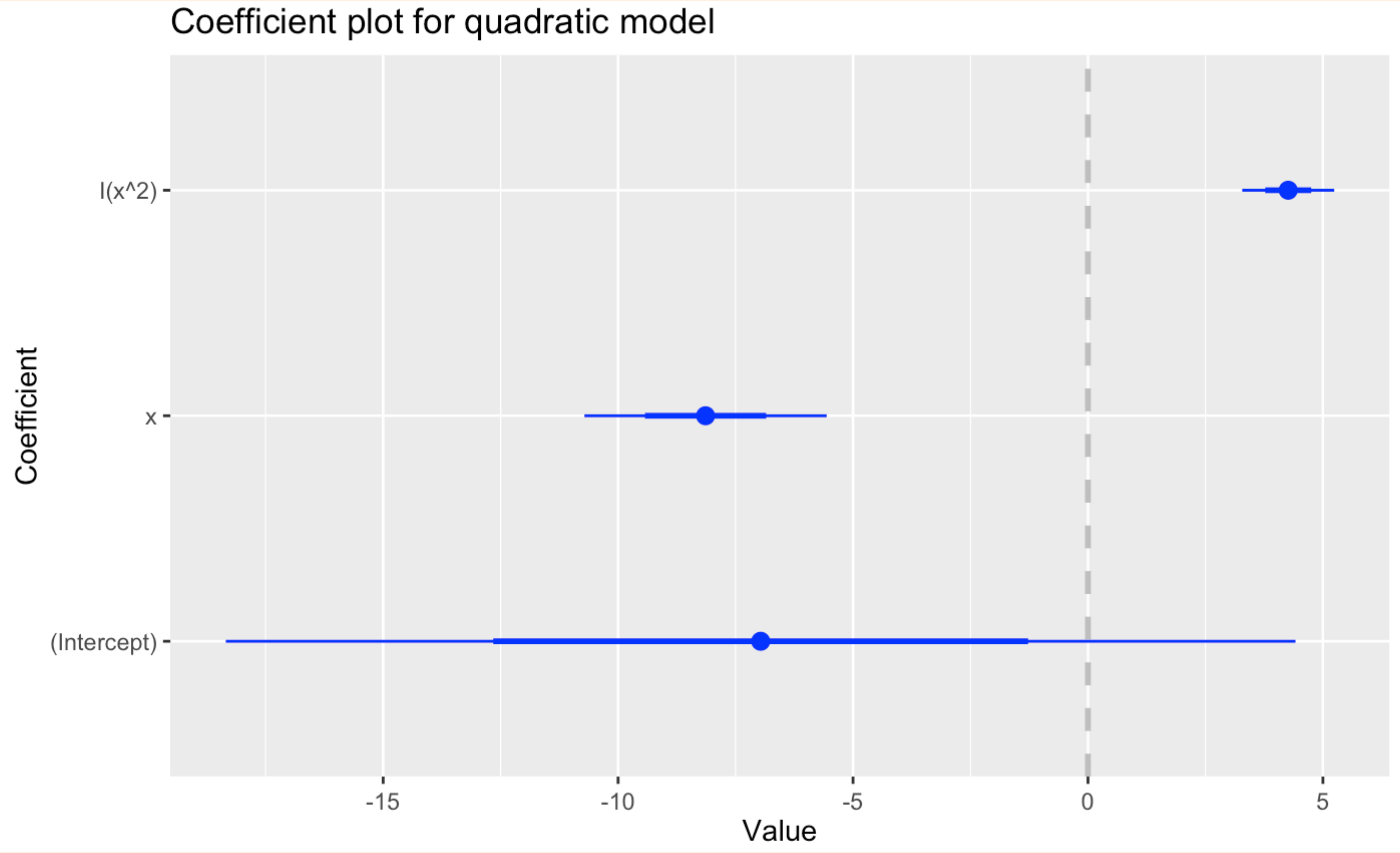
We say that a variable is statistically significant if its confidence interval does not overlap with 0

What if we wanted to try a quadratic model instead? We only need to change the formula

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \textbf{error}$$

```r
quad_mod <- lm(formula = y ~ x + I(x^2), data = df)
```

```
Call:
lm(formula = y2 ~ x + I(x^2), data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-53.898 -19.667   0.543  18.538  50.016 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)β0 -6.9655     5.6914  -1.224    0.227    
x          β1 -8.1389     1.2884  -6.317 8.21e-08 ***
I(x^2)     β2  4.2616     0.4896   8.704 1.94e-11 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 27.09 on 48 degrees of freedom
Multiple R-squared:  0.7067,	Adjusted R-squared:  0.6945 
F-statistic: 57.83 on 2 and 48 DF,  p-value: 1.64e-13
```

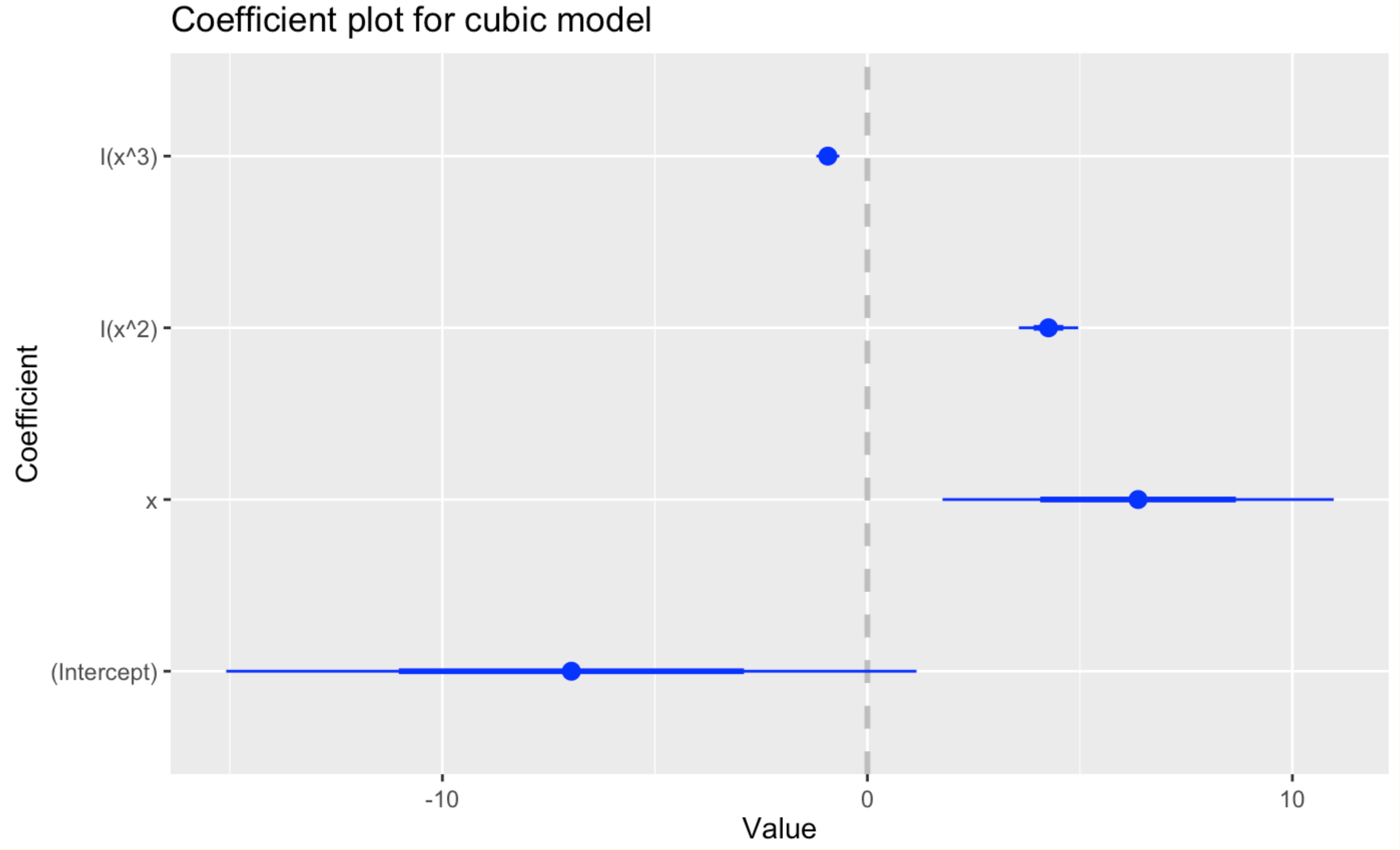


Finally, let's fit a cubic model (which we know represents the true trend)

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \textbf{error}$$

```
cubic_mod <- lm(formula = y ~ x + I(x^2) + I(x^3), data = df)
```

```
Call:
lm(formula = y2 ~ x + I(x^2) + I(x^3), data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-43.883 -16.347  -2.341  13.735  36.013 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  -6.9655     4.0607  -1.715  0.09287 .  
x             6.3693     2.3013   2.768  0.00805 ** 
I(x^2)        4.2616     0.3493  12.200 3.59e-16 ***
I(x^3)       -0.9305     0.1353  -6.877 1.26e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 19.33 on 47 degrees of freedom
Multiple R-squared:  0.8538,	Adjusted R-squared:  0.8445 
F-statistic:  91.5 on 3 and 47 DF,  p-value: < 2.2e-16
```



We know the cubic model is correct because it learns exactly the same parameters as we used to generate the data!

…but this is a toy problem, we shouldn't know the "true" values

- We need a performance metric to decide which of the models we've tested is the best for our data
- Remember we stated that the coefficients are estimated by minimizing the error.
- Specifically, the sum of squared errors between the model and the observations (this is where the phrase Least Squares comes from!)

Regression performance metrics

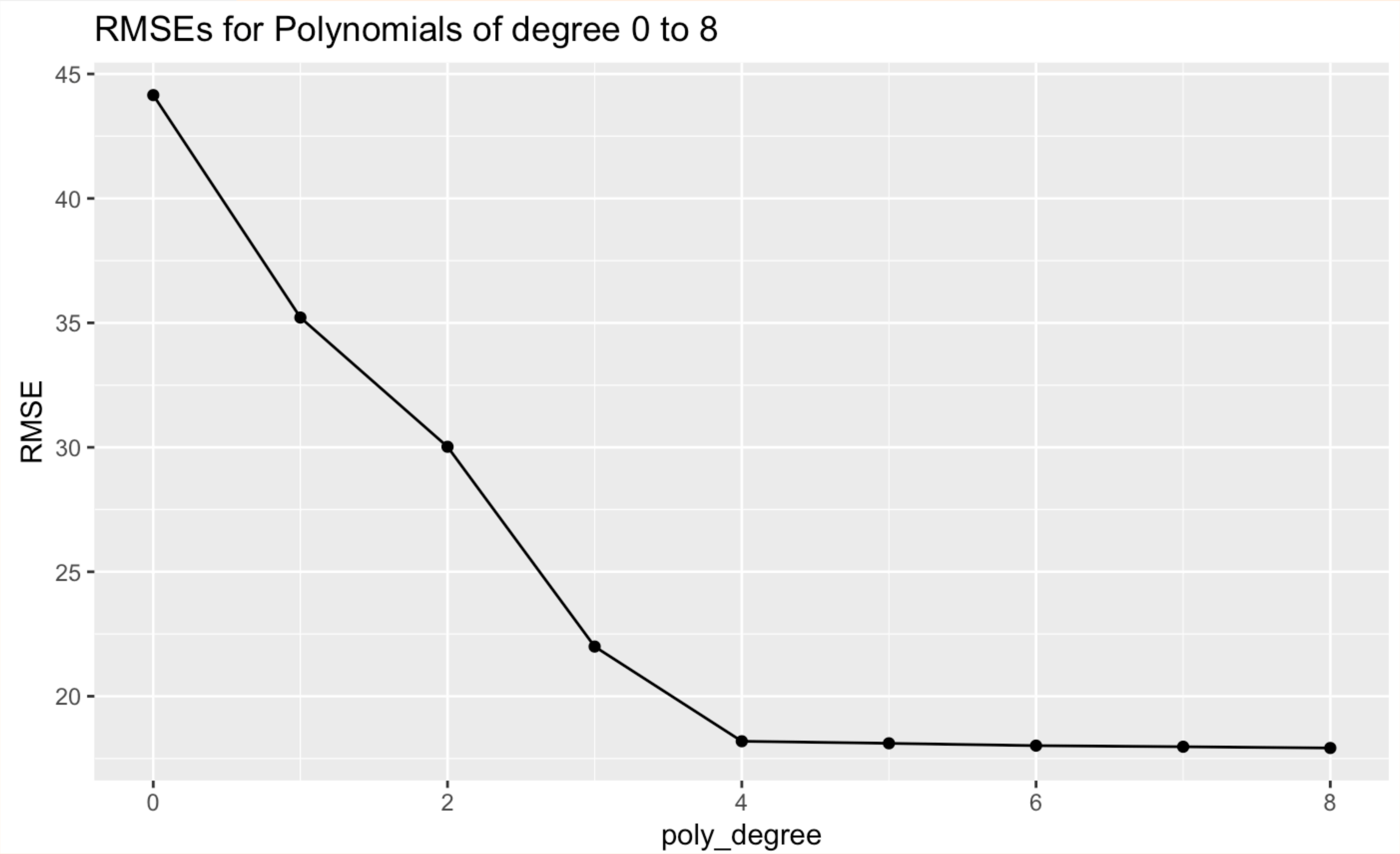
- A natural choice for the performance metric in regression problems is the Mean Squared Error (MSE).
  - The mean or average squared error across all observations.
  - The MSE is not in the same units as the response,
- Take the square root of the MSE to put the performance metric in the same units as the response
- So, it is common to consider the square Root Mean Squared Error (RMSE) as a performance metric.
- Alternatively, we could also consider the Mean Absolute Error (MAE).

Why should we stop at a degree three polynomial?

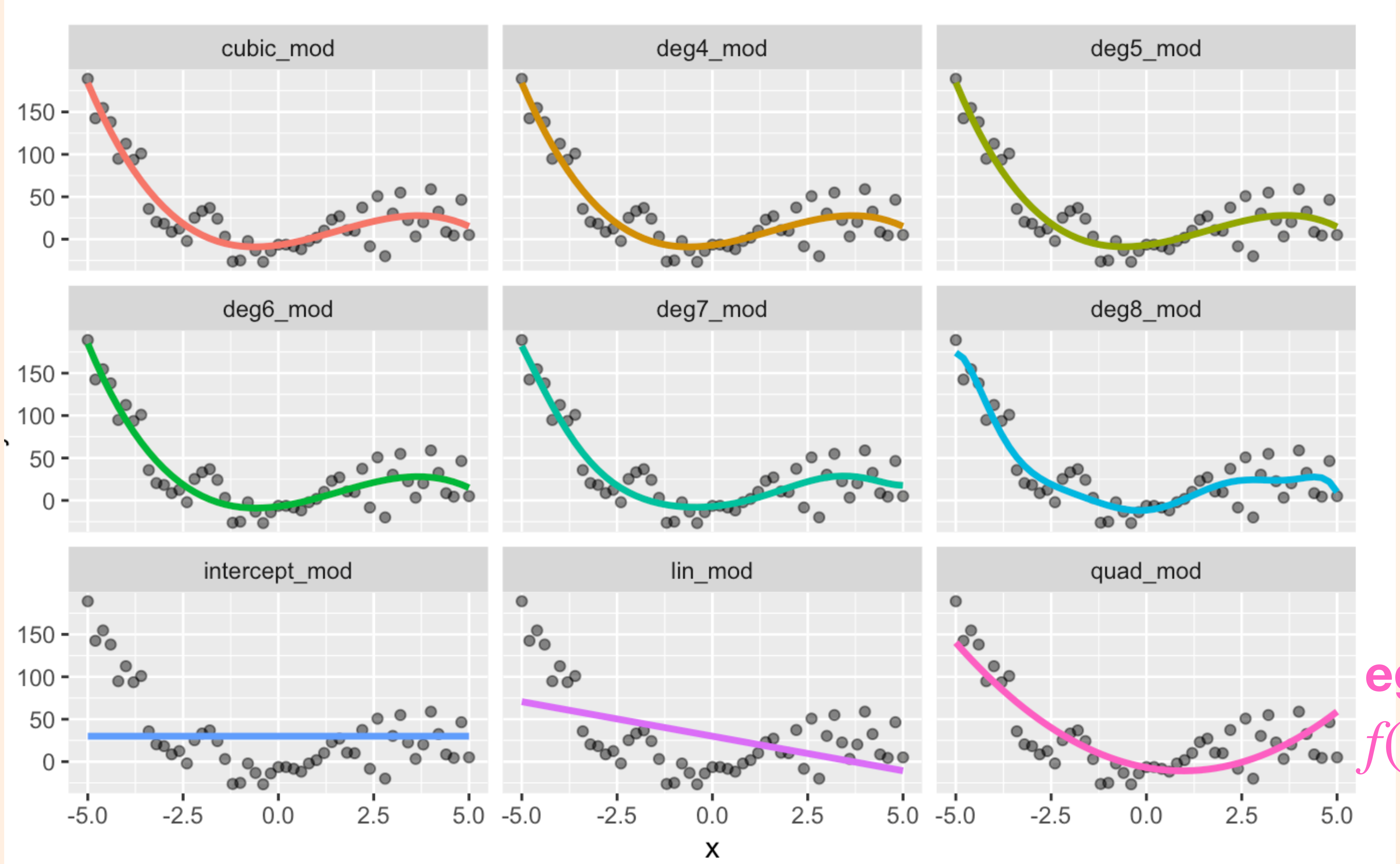
- If we did not know the true trend, we could not be sure that a higher degree polynomial might not be a better fit to the data.
- Let’s compare a total of 9 models, from a degree 0 or intercept-only model, to an 8-degree polynomial.

Why should we stop at a degree three polynomial?

```r
intercept_mod <- lm(formula = y ~ x, data = df)
deg4_mod <- lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4), data = df)
deg5_mod <- lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5), data = df)
deg6_mod <- lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6), data = df)
deg7_mod <- lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7), data = df)
deg8_mod <- lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8), data = df)

model_rmses <- purrr::map2_dfr(list(intercept_mod, lin_mod, quad_mod, cubic_mod,
                     deg4_mod, deg5_mod, deg6_mod, deg7_mod, deg8_mod),
                0:8,
                function(m, p, data_use){
                  list(poly_degree = p,
                       RMSE = modelr::rmse(m, data_use))
                },
                data_use = df)

model_rmses |> ggplot(aes(x=poly_degree,y=RMSE)) +
  geom_point() +
  geom_line() +
  labs(title="RMSEs for Polynomials of degree 0 to 8")
```



Visualize the predictive trends for all models

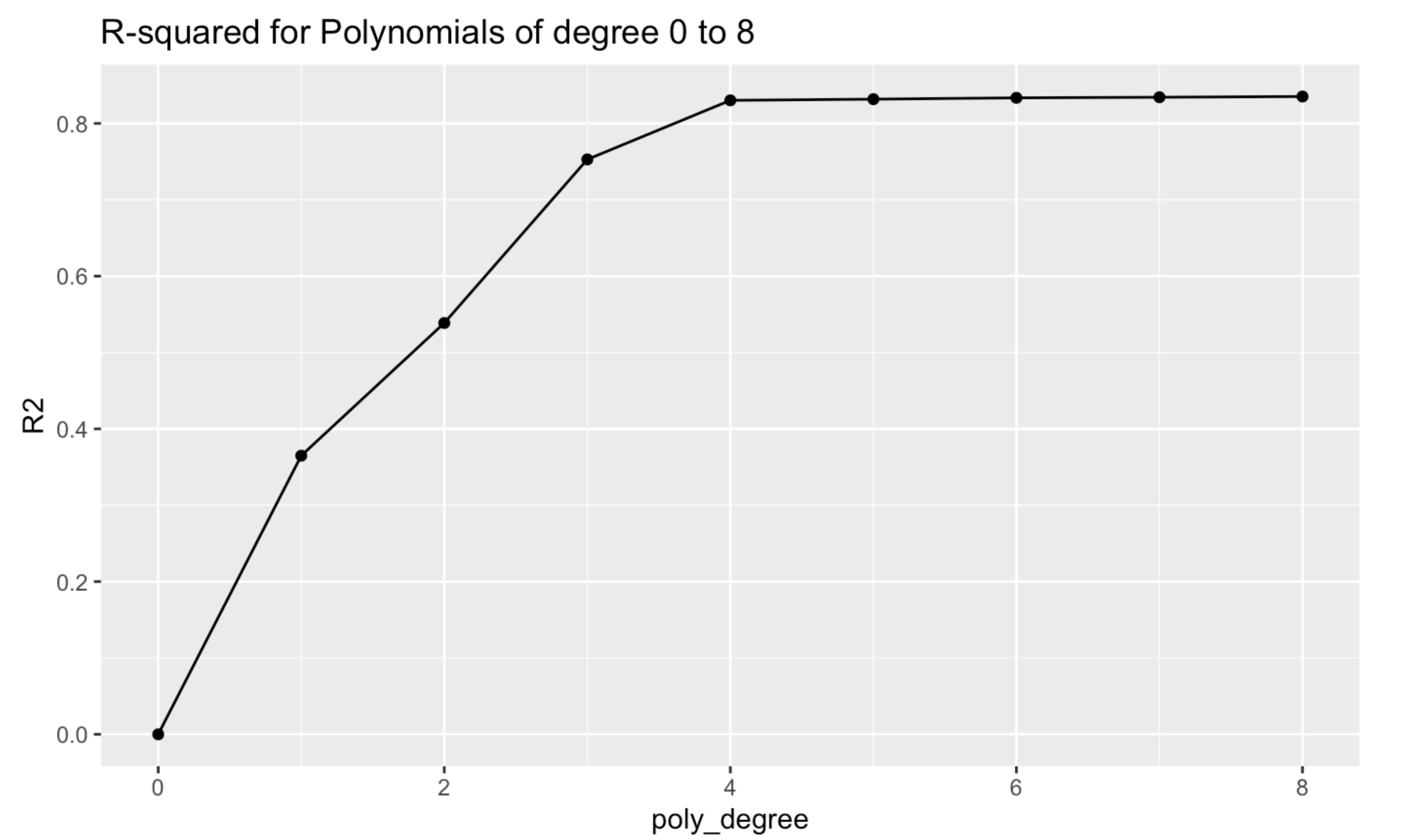


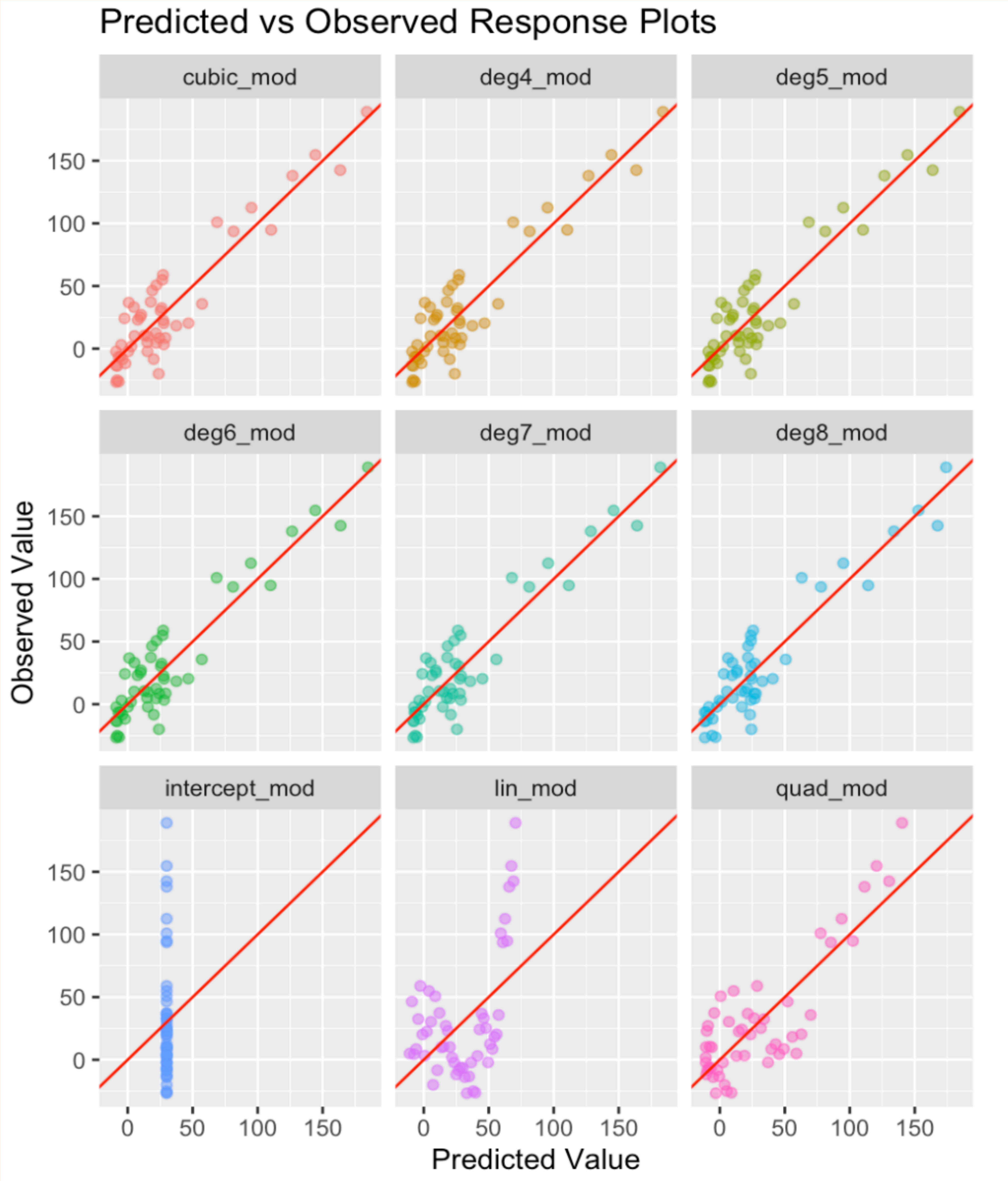
What function does each model and its learned coefficients represent?

eg quad_mod:

$$f(x) = -7 - 8.1x + 4.3x^2$$

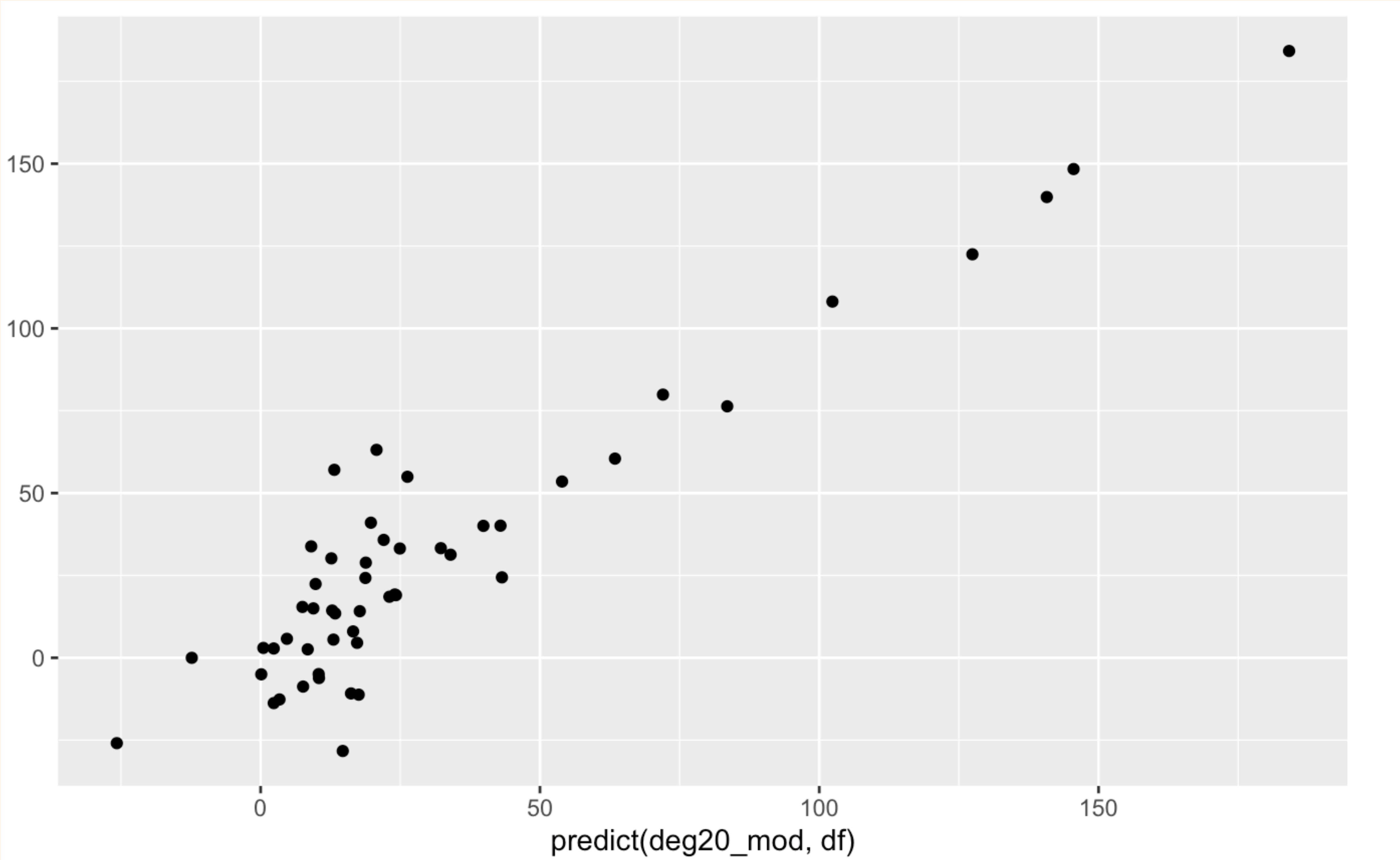
How well do the predicted values correlate with the true responses?





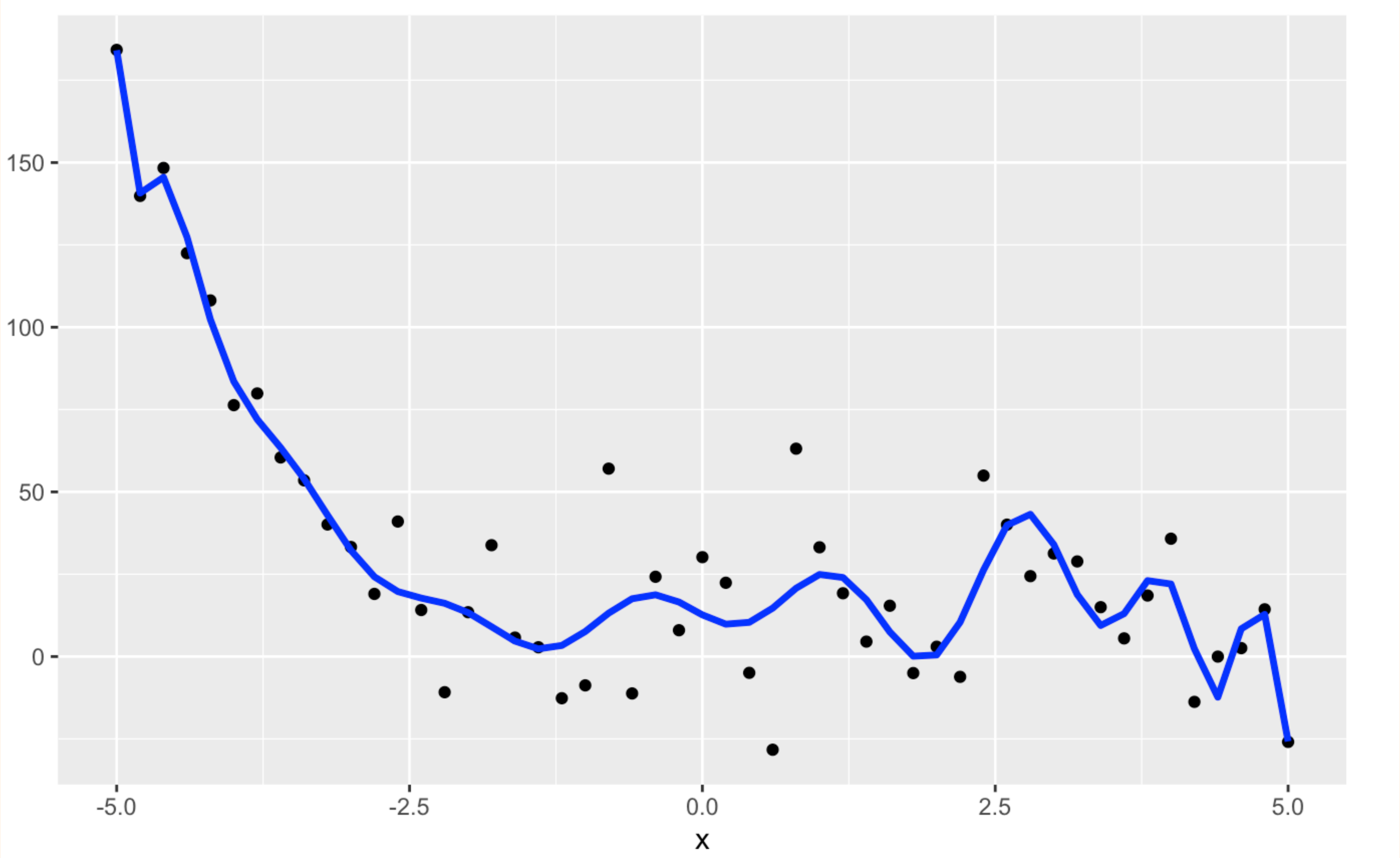
- **The R^2 performance metric measures the correlation coefficient between predicted and observed responses**

What happens to these performance metrics as the model complexity increases? Try a degree 20 polynomial



The predicted and observed responses are still highly correlated, and R2 is slightly higher than the smaller models…

What happens to these performance metrics as the model complexity increases? Try a degree 20 polynomial



But the pattern we fit is highly variable, and seems to be fitting to the noise instead of the true signal!