Class 26: Linear Modeling II

Dr. Glasbrenner

November 29, 2017

Basic method for linear fitting in R

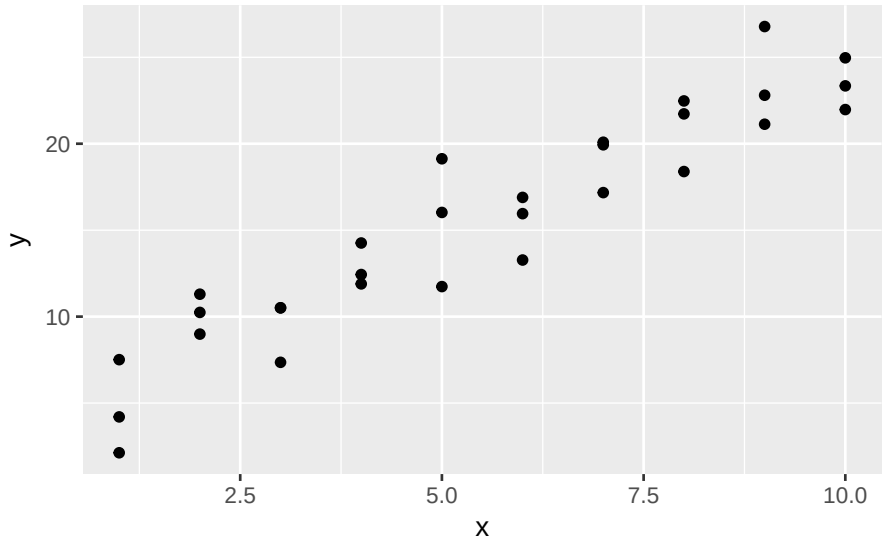
We use the `sim1` dataset loaded via `library(modelr)` for the following demonstration. First, we begin by looking at the first few rows in the dataset.

```
head(sim1)
```

```
## # A tibble: 6 x 2
##       x         y
##   <int>     <dbl>
## 1     1  4.199913
## 2     1  7.510634
## 3     1  2.125473
## 4     2  8.988857
## 5     2 10.243105
## 6     2 11.296823
```

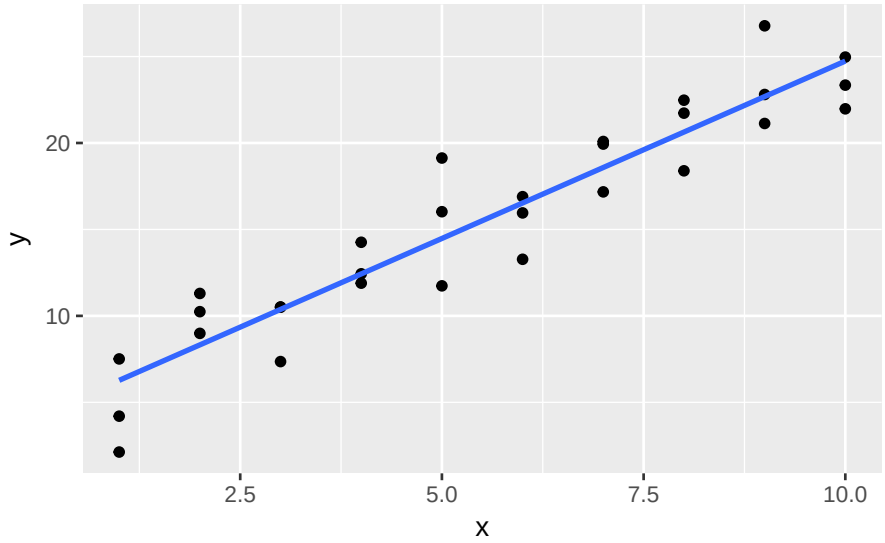
Plot the dataset

```
ggplot(sim1) +
  geom_point(aes(x, y))
```



Linear model via `ggplot2`:

```
ggplot(sim1) +
  geom_point(aes(x, y)) +
  geom_smooth(aes(x, y), method = "lm", se = FALSE)
```



Linear model via the `lm()` function:

```
sim1_mod <- lm(y ~ x, data = sim1)
```

Basic summary information

```
summary(sim1_mod)
```

```
##
## Call:
## lm(formula = y ~ x, data = sim1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1469 -1.5197  0.1331  1.4670  4.6516
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.2208     0.8688   4.858 4.09e-05 ***
## x             2.0515     0.1400  14.651 1.17e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.203 on 28 degrees of freedom
## Multiple R-squared:  0.8846, Adjusted R-squared:  0.8805
## F-statistic: 214.7 on 1 and 28 DF,  p-value: 1.173e-14
```

We report the model as:

$$y = 2.0515x + 4.2208$$

Systematic method for plotting fitted data

Should note that this method is more general, meaning it can be used even if you don't use `lm()`.

Create a series of x values with `data_grid()`:

```
sim1
```

```
## # A tibble: 30 x 2
##        x         y
##    <int>     <dbl>
##  1     1  4.199913
##  2     1  7.510634
##  3     1  2.125473
##  4     2  8.988857
##  5     2 10.243105
##  6     2 11.296823
##  7     3  7.356365
##  8     3 10.505349
##  9     3 10.511601
## 10     4 12.434589
## # ... with 20 more rows
```

```
grid <- data_grid(sim1, x)
grid
```

```
## # A tibble: 10 x 1
##        x
##    <int>
##  1     1
##  2     2
##  3     3
##  4     4
##  5     5
##  6     6
##  7     7
##  8     8
##  9     9
## 10    10
```

Use `add_predictions()` to import predictions into your tibble

```
grid2 <- add_predictions(grid, sim1_mod)
grid2
```

```
## # A tibble: 10 x 2
##        x      pred
##    <int>     <dbl>
##  1     1  6.272355
##  2     2  8.323888
##  3     3 10.375421
##  4     4 12.426954
##  5     5 14.478487
##  6     6 16.530020
##  7     7 18.581553
##  8     8 20.633087
##  9     9 22.684620
## 10    10 24.736153
```

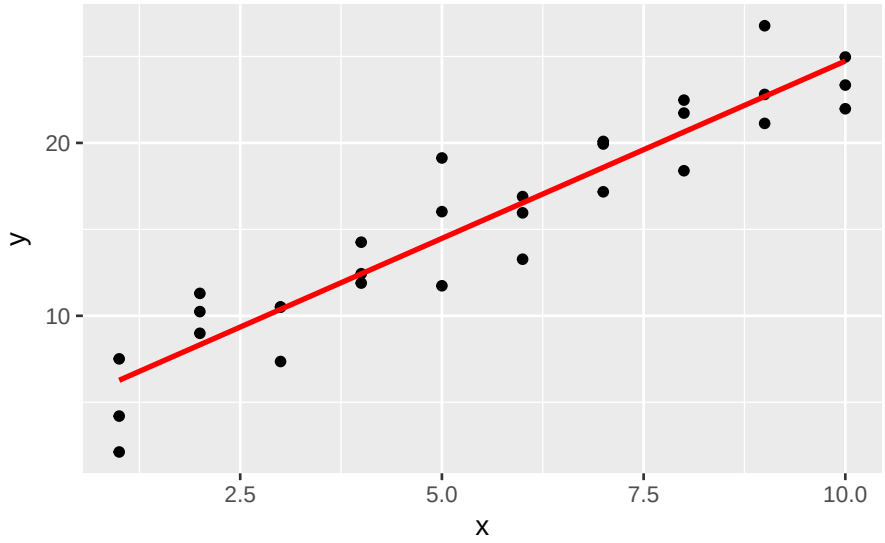
Use `add_residuals()` to extract the residuals from your fit.

```
sim1_resid <- add_residuals(sim1, sim1_mod)
sim1_resid
```

```
## # A tibble: 30 x 3
##        x         y        resid
##    <int>     <dbl>        <dbl>
##  1     1  4.199913 -2.072442018
##  2     1  7.510634  1.238279125
##  3     1  2.125473 -4.146882207
##  4     2  8.988857  0.664969362
##  5     2 10.243105  1.919217378
##  6     2 11.296823  2.972935148
##  7     3  7.356365 -3.019056466
##  8     3 10.505349  0.129928252
##  9     3 10.511601  0.136179642
## 10     4 12.434589  0.007634878
## # ... with 20 more rows
```

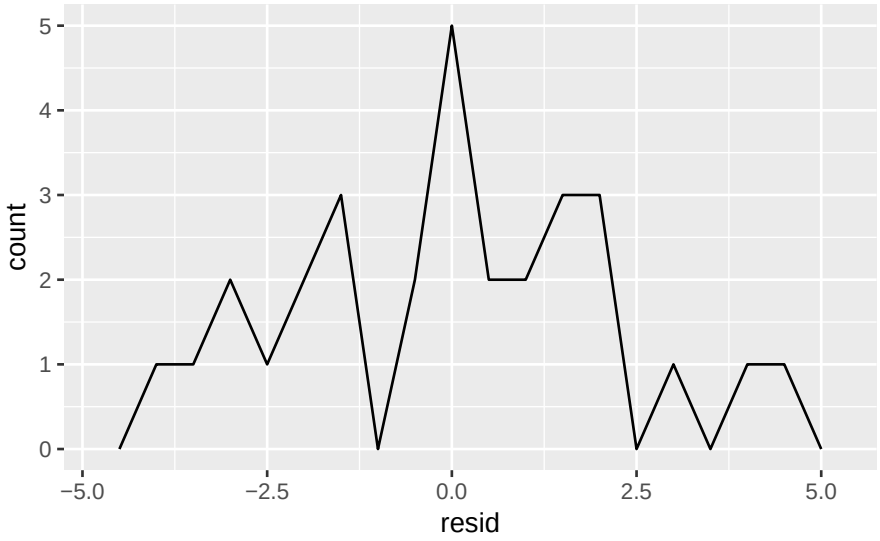
Create a plot:

```
ggplot(sim1) +
  geom_point(aes(x = x, y = y)) +
  geom_line(aes(x = x, y = pred), data = grid2, color = "red", size = 1)
```

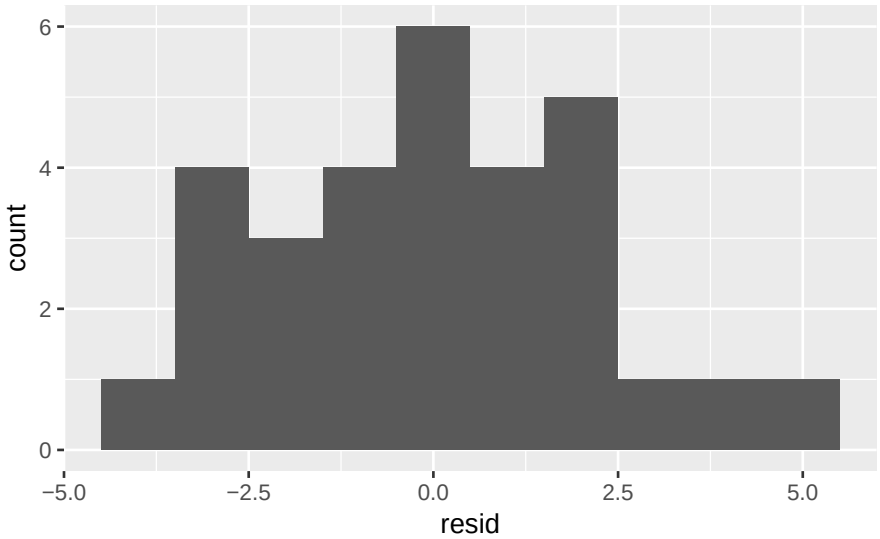


Use `geom_freqpoly()` or `geom_histogram()` to inspect the absolute residuals.

```
ggplot(sim1_resid) +
  geom_freqpoly(aes(x = resid), binwidth = 0.5)
```

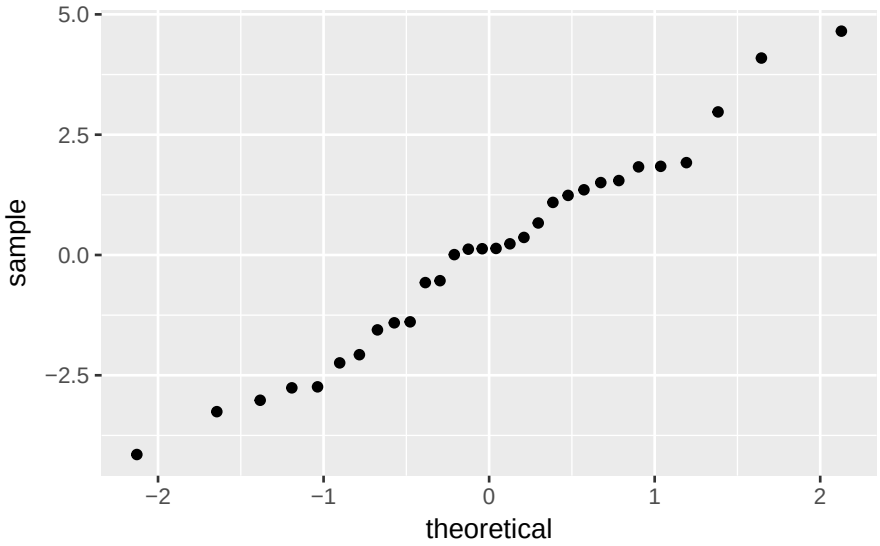



```
ggplot(sim1_resid) +
  geom_histogram(aes(x = resid), binwidth = 1)
```



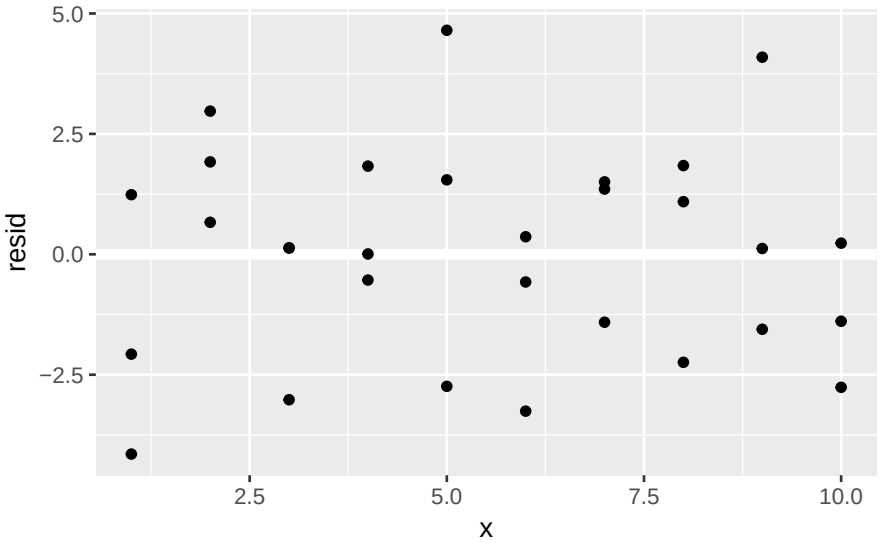
An even better test for normal residuals is a Q-Q plot:

```
ggplot(sim1_resid) +
  geom_qq(aes(sample = resid))
```



Finally, we should inspect the residual spread as a function of x to check whether the variability is constant or not:

```
ggplot(sim1_resid) +
  geom_ref_line(h = 0) +
  geom_point(aes(x = x, y = resid))
```



Practice

Use the mpg dataset and create a linear model for hwy (response variable, y axis) versus `displ` (explanatory variable, x axis):

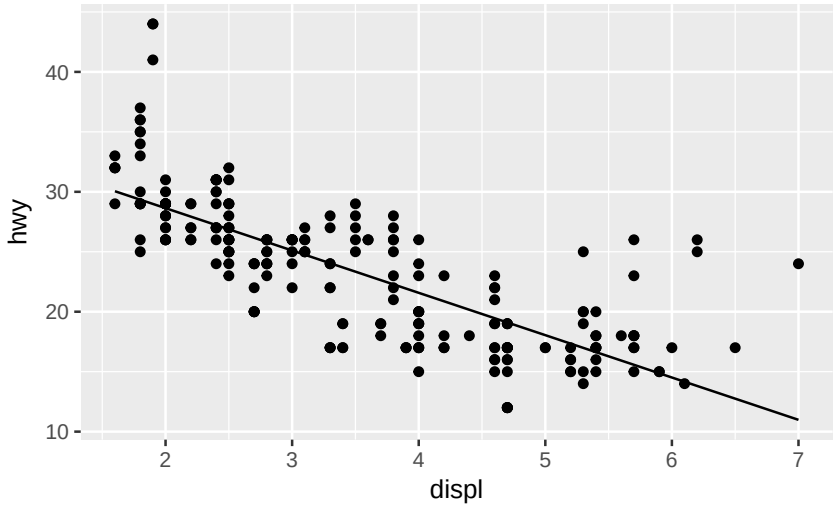
1. Create a model using `lm()`

   ```
   mpg_mod <- lm(hwy ~ displ, data = mpg)
   ```

2. Plot the points with the model

   ```
   grid_mpg <- data_grid(mpg, displ)
   grid_mpg2 <- add_predictions(grid_mpg, mpg_mod)
   ```

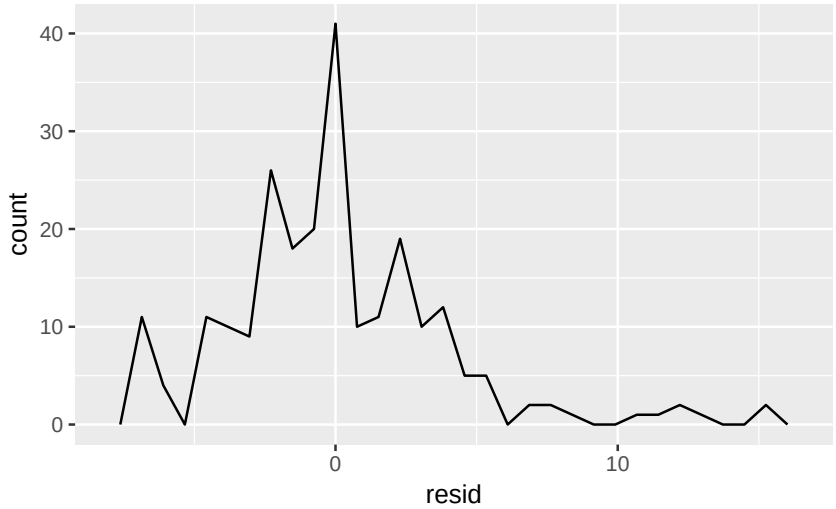
```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  geom_line(aes(x = displ, y = pred), data = grid_mpg2)
```



3. Plot the residuals

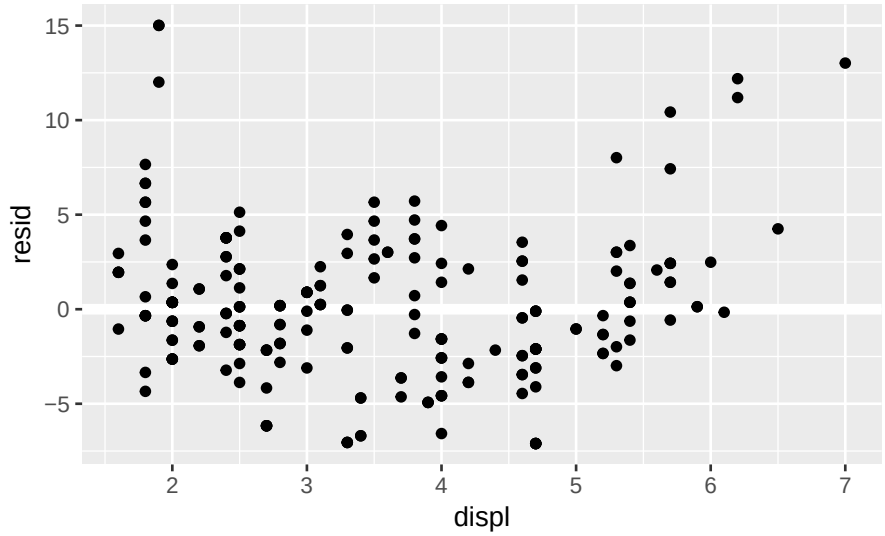
```
mpg_resid <- add_residuals(mpg, mpg_mod)

ggplot(mpg_resid) +
  geom_freqpoly(aes(x = resid))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We also should check the variability of the residuals as a function of `displ`:

```
ggplot(mpg_resid) +
  geom_ref_line(h = 0) +
  geom_point(aes(x = displ, y = resid))
```



The bending curvature is evidence that a linear model is not an appropriate model for this dataset.

Non-linear one-term modeling

Nonlinear one-term modeling:

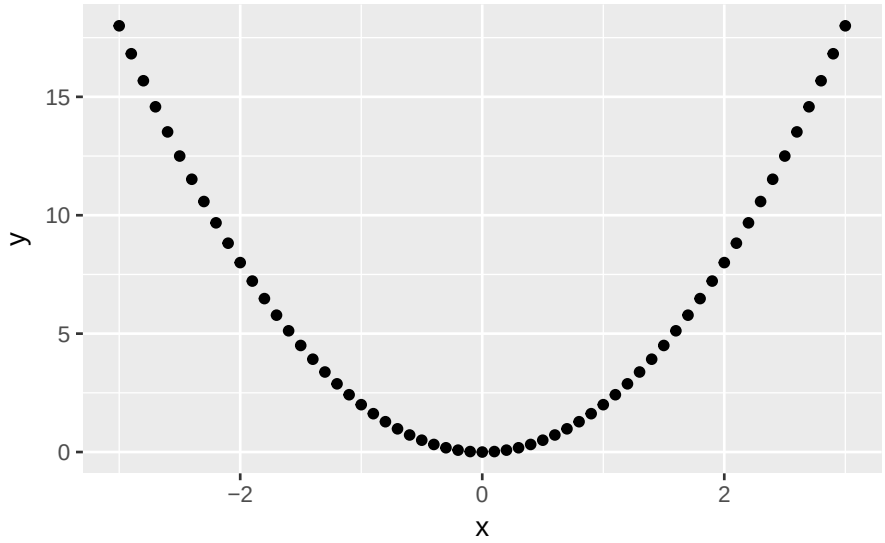
$$y = 2x^2$$

```
x1 <- seq(-3, 3, 0.1)
y1 <- 2 * x1^2
parabola <- tibble(x = x1, y = y1)
parabola
```

```
## # A tibble: 61 x 2
##        x     y
##    <dbl> <dbl>
##  1  -3.0 18.00
##  2  -2.9 16.82
##  3  -2.8 15.68
##  4  -2.7 14.58
##  5  -2.6 13.52
##  6  -2.5 12.50
##  7  -2.4 11.52
##  8  -2.3 10.58
##  9  -2.2  9.68
## 10  -2.1  8.82
## # ... with 51 more rows
```

Visualize the parabola data:

```
ggplot(parabola) +
  geom_point(aes(x, y))
```



You can actually fit this using `lm()`, but you need to be careful how you specify the x^2 part. For example, this **doesn't** work:

```
parabola_mod1 <- lm(y ~ x^2, data = parabola)
summary(parabola_mod1)
```

```
##
## Call:
## lm(formula = y ~ x^2, data = parabola)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -6.20  -4.92  -1.70   4.38  11.80
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 6.200e+00  7.217e-01   8.591 5.52e-12 ***
## x           9.042e-16  4.099e-01   0.000        1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.636 on 59 degrees of freedom
## Multiple R-squared:  1.759e-31,  Adjusted R-squared:  -0.01695
## F-statistic: 1.038e-29 on 1 and 59 DF,  p-value: 1
```

If you warp `x^2` with the `I()` function, then we get the expected behavior:

```
parabola_mod2 <- lm(y ~ I(x^2), data = parabola)
summary(parabola_mod2)
```

```
## Warning in summary.lm(parabola_mod2): essentially perfect fit: summary may
## be unreliable
```

```
##
## Call:
## lm(formula = y ~ I(x^2), data = parabola)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
```

```
## -3.461e-14 -3.180e-16  2.050e-16  1.131e-15  7.017e-15
##
## Coefficients:
##              Estimate Std. Error   t value Pr(>|t|)
## (Intercept) 4.549e-15  9.081e-16 5.009e+00 5.27e-06 ***
## I(x^2)      2.000e+00  2.184e-16 9.158e+15  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.727e-15 on 59 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 8.388e+31 on 1 and 59 DF,  p-value: < 2.2e-16
```

Why would this be necessary? The following quote from *R for Data Science* explains:

> You can also perform transformations inside the model formula. For example, `log(y) ~ sqrt(x1) + x2` is transformed to `log(y) = a_1 + a_2 * sqrt(x1) + a_3 * x2`. If your transformation involves +, *, ^, or -, you'll need to wrap it in `I()` so R doesn't treat it like part of the model specification. For example, `y ~ x + I(x ^ 2)` is translated to `y = a_1 + a_2 * x + a_3 * x^2`. If you forget the `I()` and specify `y ~ x ^ 2 + x`, R will compute `y ~ x * x + x`. `x * x` means the interaction of `x` with itself, which is the same as `x`. R automatically drops redundant variables so `x + x` become `x`, meaning that `y ~ x ^ 2 + x` specifies the function `y = a_1 + a_2 * x`. That's probably not what you intended!