L14 Model Basics

Data Science I (STAT 301-1)

YOUR NAME

Contents

Overview

The goal of this lab is to use modeling to explore data. The goal of a model is to provide a simple low-dimensional summary of a dataset. Ideally, the model will capture true "signals" and ignore "noise". Models are an important tool for exploration and, as such, the tidyverse supplies the package `modelr` to streamline the process. Note that `modelr` is not part of the core tidyverse and will need to be installed separately.

Datasets

Datasets referenced in this lab are from the tidyverse packages and/or are referenced in the Model Basics section of our textbook.

Exercises

Please complete the following exercises. Be sure that your solutions are clearly indicated and that the document is neatly formatted.

```r
# Loading package(s)
library(tidyverse)
library(modelr)
options(na.action = na.warn)
```

Load Packages

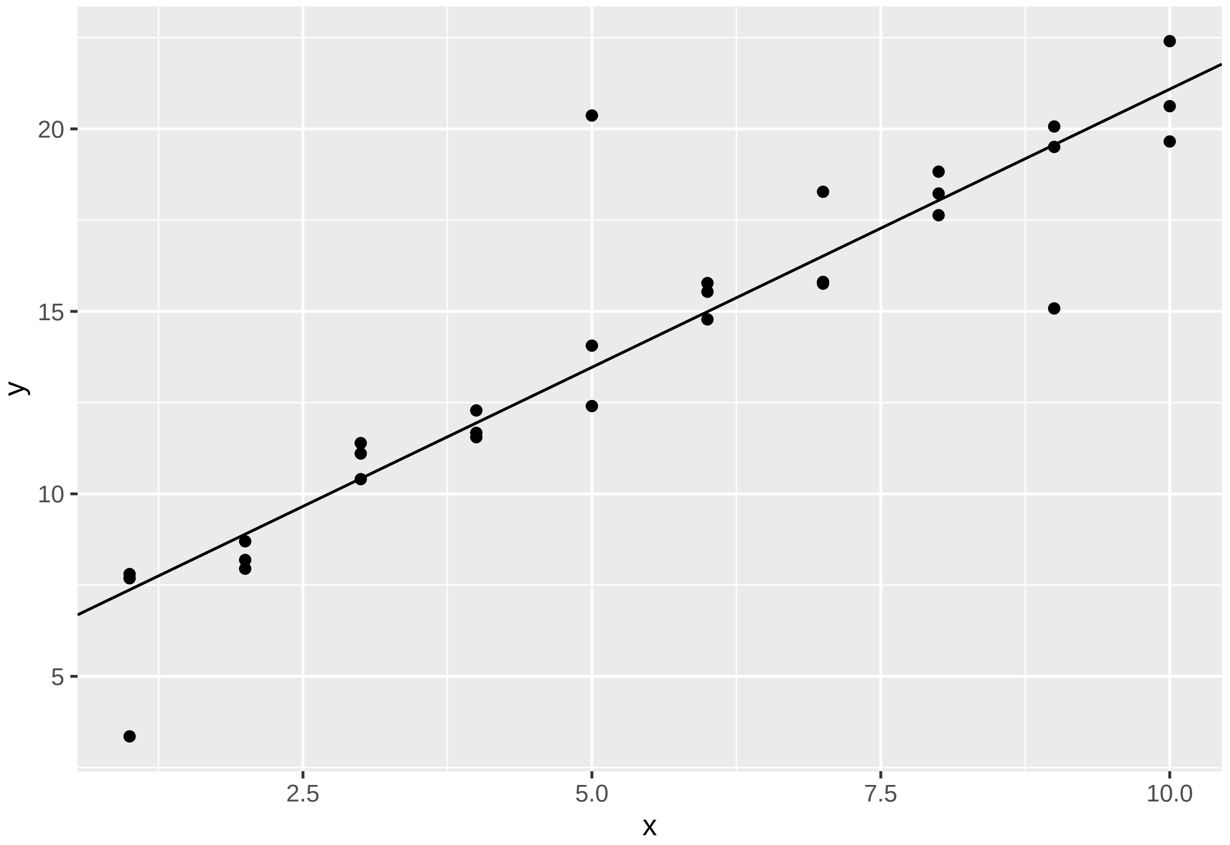
Exercise 1 (Website: 23.2.1 Ex. 1)

One downside of the linear model is that it is sensitive to unusual values because the distance incorporates a squared term. Fit a linear model to the simulated data below, and visualize the results. Rerun a few times to generate different simulated datasets. What do you notice about the model?

```r
sim1a <- tibble(
  x = rep(1:10, each = 3),
  y = x * 1.5 + 6 + rt(length(x), df = 2)
)

sim_and_plot <- function() {
  sim1a_mod <- lm(y ~ x, data = sim1a)
  ggplot(data = sim1a) +
    geom_point(mapping = aes(x, y)) +
    geom_abline(aes(intercept = sim1a_mod$coef[1], slope = sim1a_mod$coef[2]))
}

sim_and_plot()
```



The model leaves outliers in, which can cause large bias. It appears that significant outliers are fairly likely in this simulation, which because of the squared term, drastically throws off the model.

Exercise 2 (Website: 23.2.1 Ex. 2)

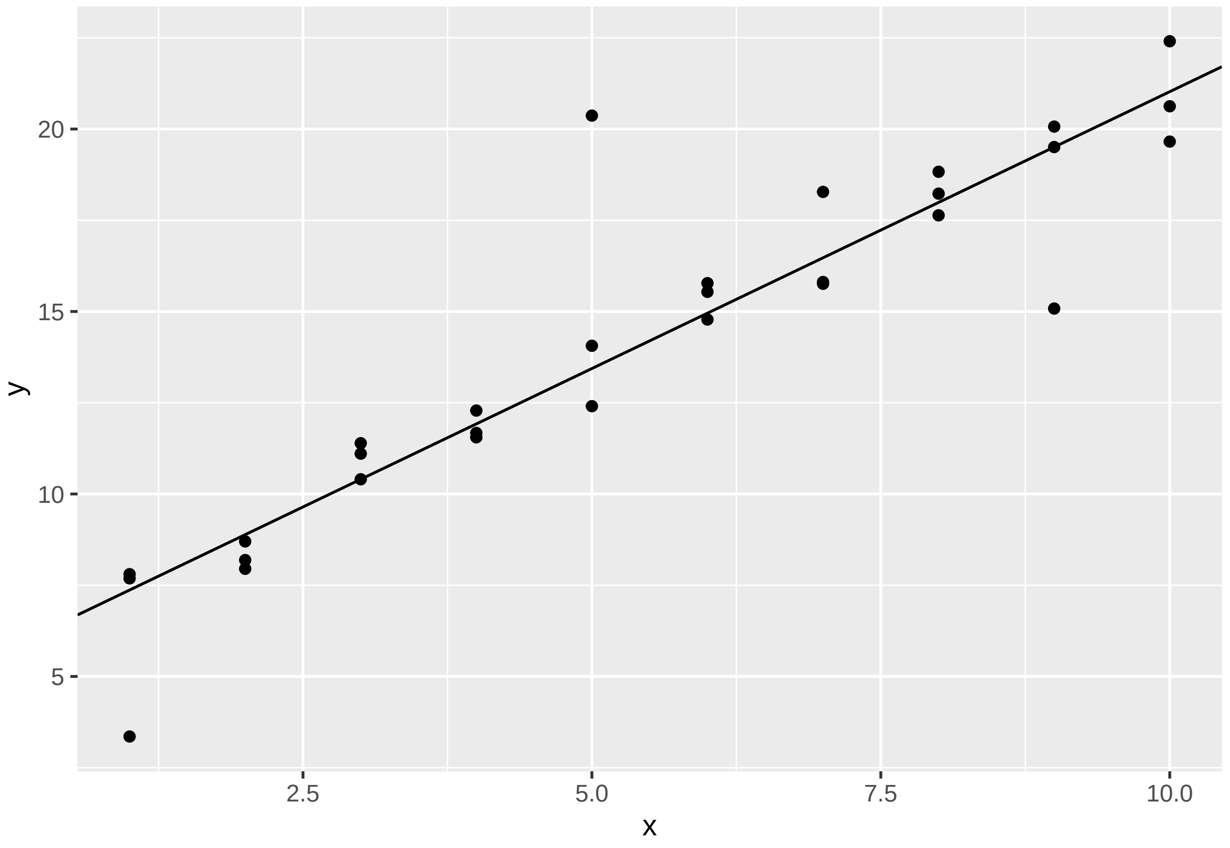
One way to make linear models more robust is to use a different distance measure. For example, instead of root-mean-squared distance, you could use mean-absolute distance. Use `optim()` to fit this model to the simulated data above and compare its results to those of the linear model.

```r
measure_distance <- function(mod, data) {
  diff <- data$y - make_prediction(mod, data)
  mean(abs(diff))
}


make_prediction <- function(mod, data) {
  mod[1] + mod[2] * data$x
}

best <- optim(c(0, 0), measure_distance, data = sim1a)

ggplot(sim1a, aes(x, y)) +
  geom_point() +
  geom_abline(intercept = best$par[1], slope = best$par[2])
```



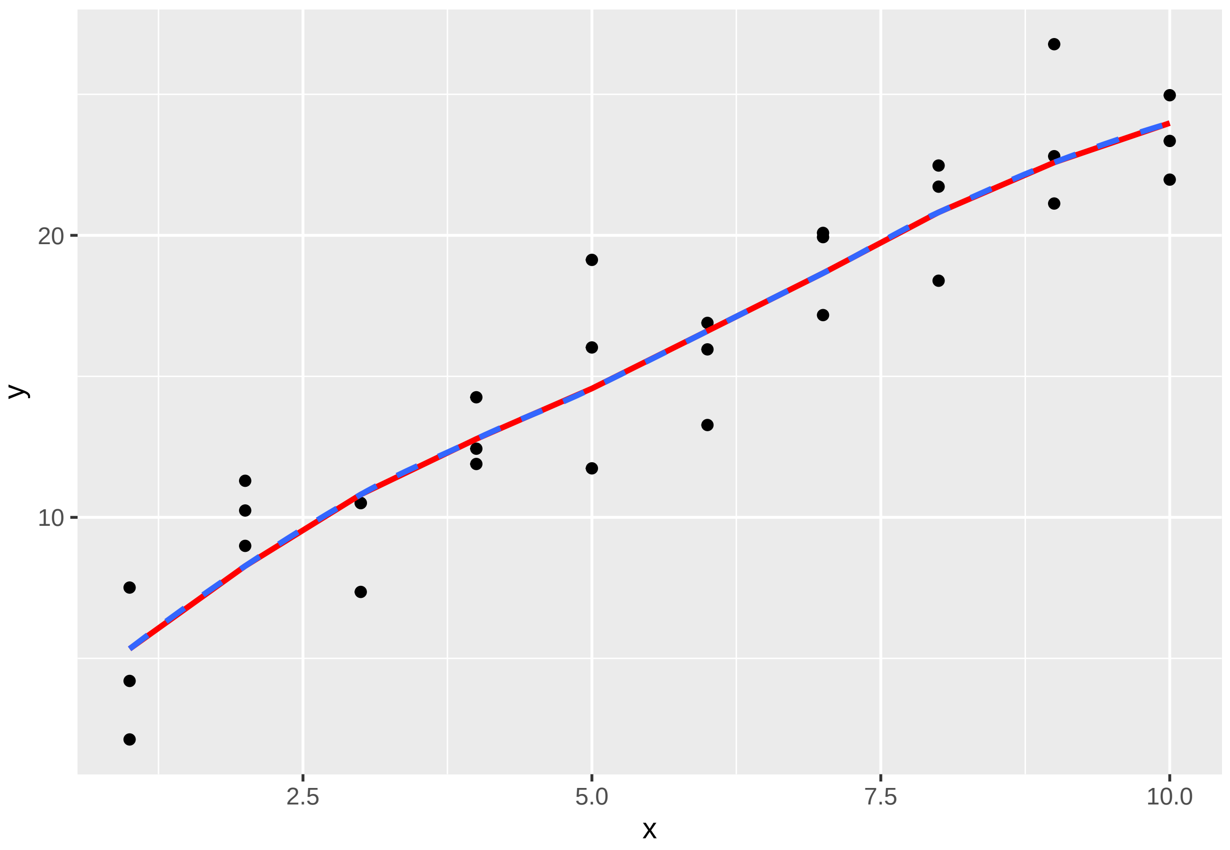
It looks like the `optim()` method does a better job of accounting for outliers than the linear model.

Exercise 3 (Website: 23.3.3 Ex. 1)

Instead of using `lm()` to fit a straight line, you can use `loess()` to fit a smooth curve. Repeat the process of model fitting, grid generation, predictions, and visualization on `sim1` using `loess()` instead of `lm()`. How does the result compare to `geom_smooth()`?

```
grid <- sim1 %>%
  data_grid(x)
sim1_loess <- loess(y ~ x, data = sim1)
grid <- grid %>%
  add_predictions(sim1_loess)
```

```
ggplot(sim1, aes(x)) +
  geom_point(aes(y = y)) +
  geom_line(aes(y = pred), data = grid, colour = "red", size = 1) +
  geom_smooth(aes(x, y), se = FALSE, linetype = 'dashed')
```



The `loess` line-fitting method and the `geom_smooth()` function give the exact same output. In fact, `geom_smooth()` uses `loess` to calculate the smooth line.

Exercise 4 (Website: 23.3.3 Ex. 2)

`add_predictions()` is paired with `gather_predictions()` and `spread_predictions()`. How do these three functions differ? **`add_prediction()` adds a single new column to the input data. `spread_predictions()` and `gather_predictions()` are used to add multiple models at once. `spread_predictions()` adds one column for each model, whereas `gather_predictions()` adds two columns, `.model` and `.pred`, for each model.**

Exercise 5 (Website: 23.4.5 Ex. 1)

Repeat the analysis of `sim2` using a model without an intercept. What happens to the model equation? What happens to the predictions?

```
sim2_int <- lm(y ~ x, data = sim2)
sim2_no_int <- lm(y ~ x - 1, data = sim2)
grid <- sim2 %>%
  data_grid(x) %>%
  spread_predictions(sim2_int, sim2_no_int)
grid
```

```
## # A tibble: 4 x 3
##   x     sim2_int sim2_no_int
##   <chr>    <dbl>       <dbl>
## 1 a         1.15        1.15
## 2 b         8.12        8.12
## 3 c         6.13        6.13
## 4 d         1.91        1.91
```

Both models are exactly the same, no matter if you have an intercept or not.

Exercise 6 (Website: 23.4.5 Ex. 2)

Use `model_matrix()` to explore the equations generated for the models fit to `sim3` and `sim4` in the book. Why is `*` a good shorthand for interaction?

```
sim3
```

```
## # A tibble: 120 x 5
##       x1 x2      rep      y    sd
##    <int> <fct> <int>  <dbl> <dbl>
##  1     1 a         1 -0.571     2
##  2     1 a         2  1.18      2
##  3     1 a         3  2.24      2
##  4     1 b         1  7.44      2
##  5     1 b         2  8.52      2
##  6     1 b         3  7.72      2
##  7     1 c         1  6.51      2
##  8     1 c         2  5.79      2
##  9     1 c         3  6.07      2
## 10     1 d         1  2.11      2
## # ... with 110 more rows
```

```
#`x1` is a numeric variable, wheras `x2`  is a categorical variable.
model_matrix(y ~ x1 * x2, data = sim3)
```

```
## # A tibble: 120 x 8
##    `(Intercept)`    x1   x2b   x2c   x2d `x1:x2b` `x1:x2c` `x1:x2d`
##            <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>    <dbl>    <dbl>
##  1             1     1     0     0     0        0        0        0
##  2             1     1     0     0     0        0        0        0
##  3             1     1     0     0     0        0        0        0
##  4             1     1     1     0     0        1        0        0
##  5             1     1     1     0     0        1        0        0
##  6             1     1     1     0     0        1        0        0
##  7             1     1     0     1     0        0        1        0
##  8             1     1     0     1     0        0        1        0
##  9             1     1     0     1     0        0        1        0
## 10             1     1     0     0     1        0        0        1
## # ... with 110 more rows
```

This creates 6 new variables, `x2b`, `x2c`, `x2d`, `x1:x2b`, `x1:x2c`, and `x1:x2d`, where the latter three are the binary product of `x1` and the specific `x2`. The reason that there is no `x2a` is because a value of 0 for `x2b`, `x2c`, and `x2d` signifies a value of 1 for `x2a`, and thus it is left out of the model. The `*` is appropriate because it is literally a multiplication (in binary terms).

```
sim4
```

```
## # A tibble: 300 x 4
##       x1     x2   rep       y
##    <dbl>  <dbl> <int>   <dbl>
##  1    -1 -1         1  4.25
##  2    -1 -1         2  1.21
##  3    -1 -1         3  0.353
##  4    -1 -0.778     1 -0.0467
##  5    -1 -0.778     2  4.64
##  6    -1 -0.778     3  1.38
##  7    -1 -0.556     1  0.975
##  8    -1 -0.556     2  2.50
##  9    -1 -0.556     3  2.70
## 10    -1 -0.333     1  0.558
## # ... with 290 more rows
```

```
#Here, both `x1` and `x2` are numerical variabls.
model_matrix(y ~ x1 * x2, data = sim4)
```

```
## # A tibble: 300 x 4
##    `(Intercept)`    x1     x2 `x1:x2`
##            <dbl> <dbl>  <dbl>   <dbl>
##  1             1    -1 -1       1
##  2             1    -1 -1       1
##  3             1    -1 -1       1
##  4             1    -1 -0.778   0.778
##  5             1    -1 -0.778   0.778
##  6             1    -1 -0.778   0.778
##  7             1    -1 -0.556   0.556
##  8             1    -1 -0.556   0.556
##  9             1    -1 -0.556   0.556
## 10             1    -1 -0.333   0.333
## # ... with 290 more rows
```

We don't need the other `x2` variables here to describe it because it is not categorical. Thus the model only creates one additional variable, `x1:x2`, which is the product of `x1` and `x2`.

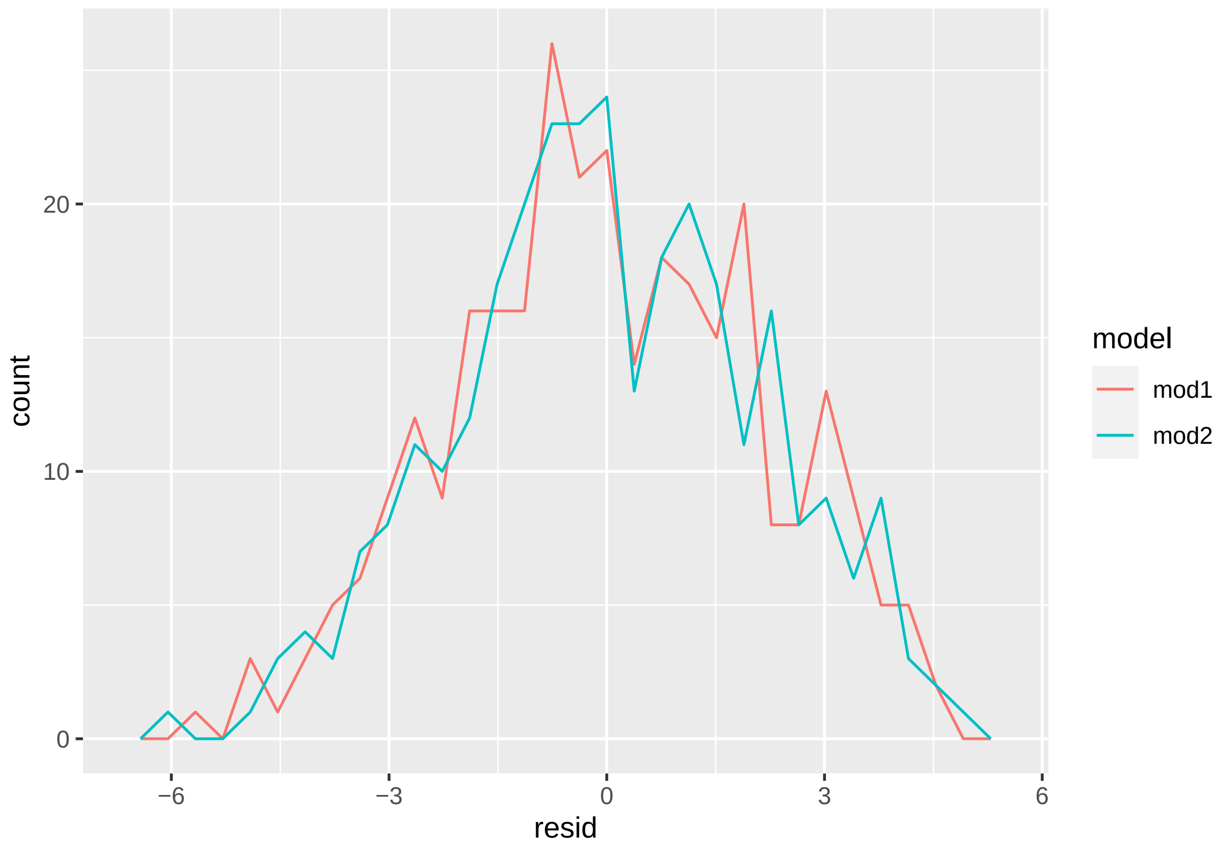
Exercise 7 (Website: 21.3.5 Ex. 1)

For `sim4`, which is better, `mod1` or `mod2`? We think `mod2` does a slightly better job at removing patterns, but it's pretty subtle. Can you come up with a plot to support the claim?

```
mod1 <- lm(y ~ x1 + x2, data = sim4)
mod2 <- lm(y ~ x1 * x2, data = sim4)
#We could look at the residuals to see what the models miss

sim4_res <- gather_residuals(sim4, mod1, mod2)

ggplot(sim4_res) +
  geom_freqpoly(mapping = aes(x = resid, color = model))
```


count
20
10
0
−6
−3
0
3
6
resid
model
mod1
mod2