R4DS HW#2

Ricardo Alexandro Aguilar

February 19, 2018

23.2.1.

#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 visualise the results. Rerun a few times to generate different simulated datasets. What do you notice about the model?

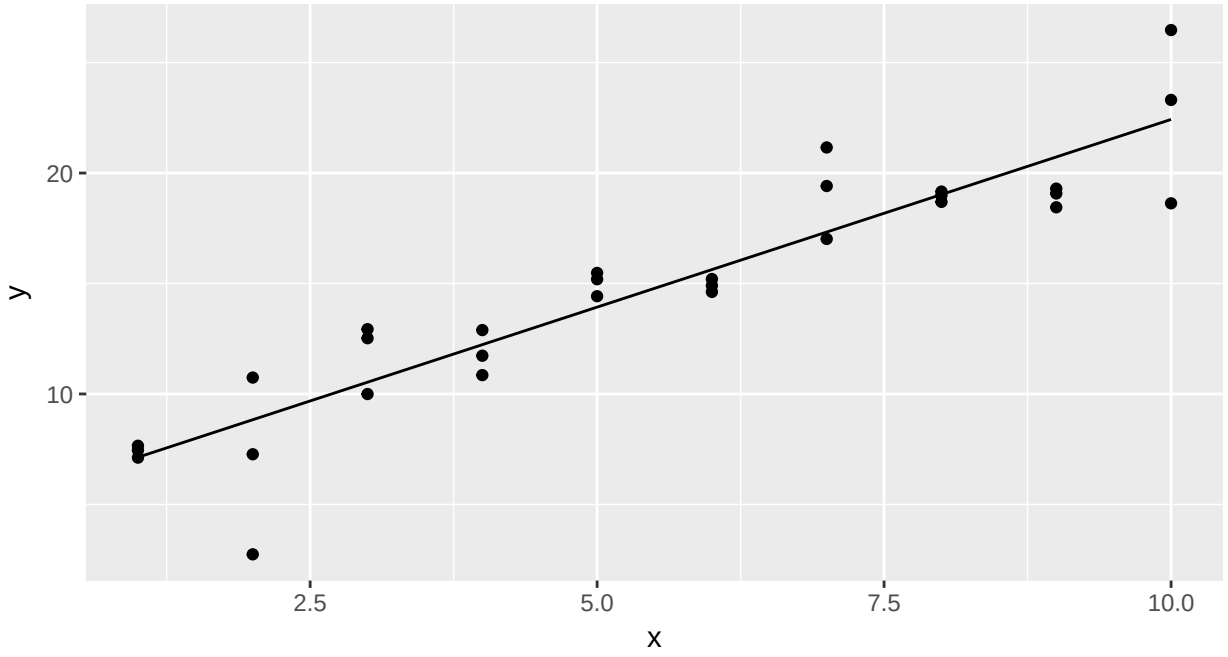
```
sim1a <- tibble(
  x = rep(1:10, each = 3),
  y = x * 1.5 + 6 + rt(length(x), df = 2)
)

model <- lm(y ~ x, data = sim1a)
pander(model)
```

Table 1: Fitting linear model: y ~ x

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	5.435	0.8262	6.578	3.916e-07
x	1.699	0.1332	12.76	3.454e-13

```
ggplot(sim1a, aes(x = x, y = y)) + geom_point() + geom_line(stat ="smooth", method = "lm")
```



Any outliers have throw the line off.

#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 it to the linear model.

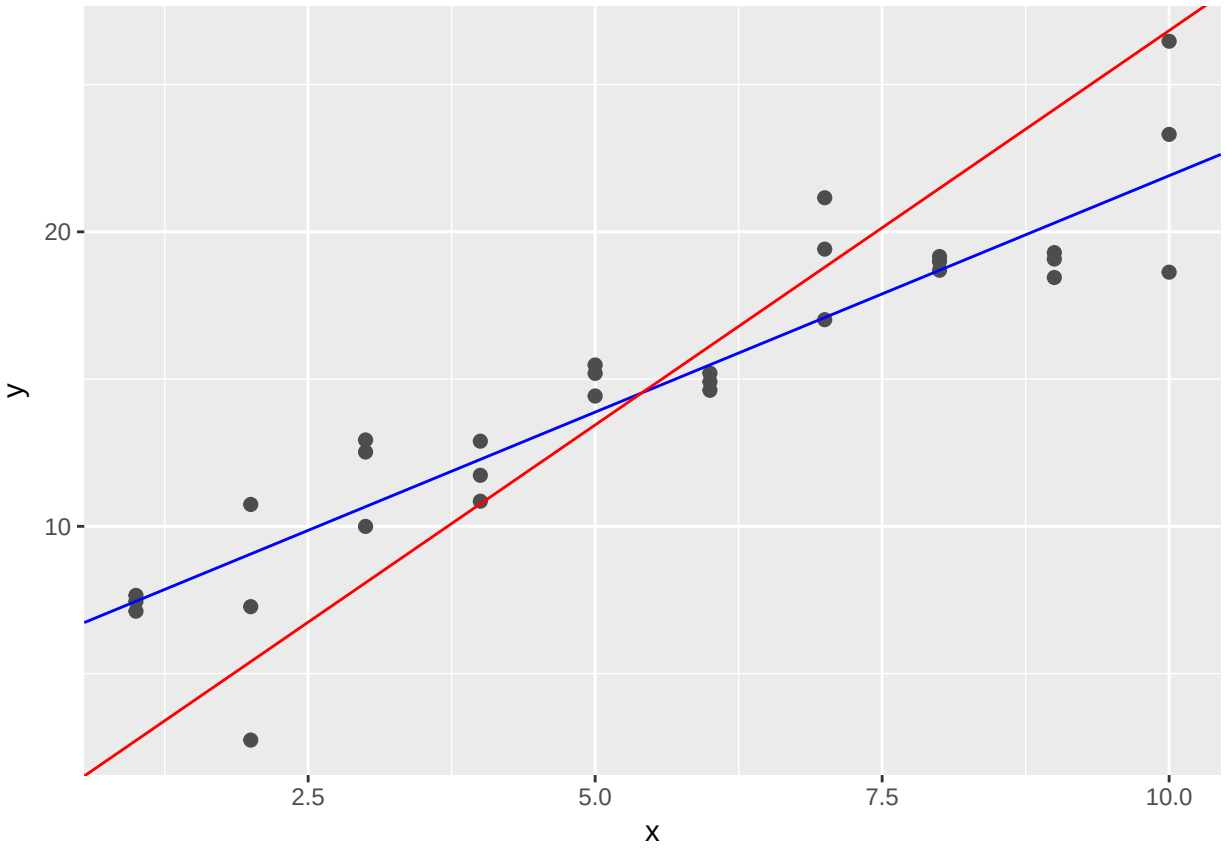
```
make_prediction <- function(a, data) {
  a[1] + data$x * a[2]
}

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

rms_distance <- function(mod, data) {
  diff <- data$y - make_prediction(mod, data)
  sqrt(mean(diff))
}

lm_ma <- optim(c(0,0), measure_distance, data = sim1a)
lm_sm <- optim(c(0,0), rms_distance, data = sim1a)

ggplot(sim1a, aes(x, y)) +
  geom_point(size = 2, colour = "grey30") +
  geom_abline(intercept = lm_ma$par[1], slope = lm_ma$par[2], color = "blue") +
  geom_abline(intercept = lm_sm$par[1], slope = lm_sm$par[2], color = "red")
```



The model that uses the mean-absolute distance (blue) is clearly not affected as much, by outliers, as the line that uses root-mean-squared distance.

23.3.3

#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 visualisation on sim1 using loess() instead of lm(). How does the result compare to geom_smooth()?

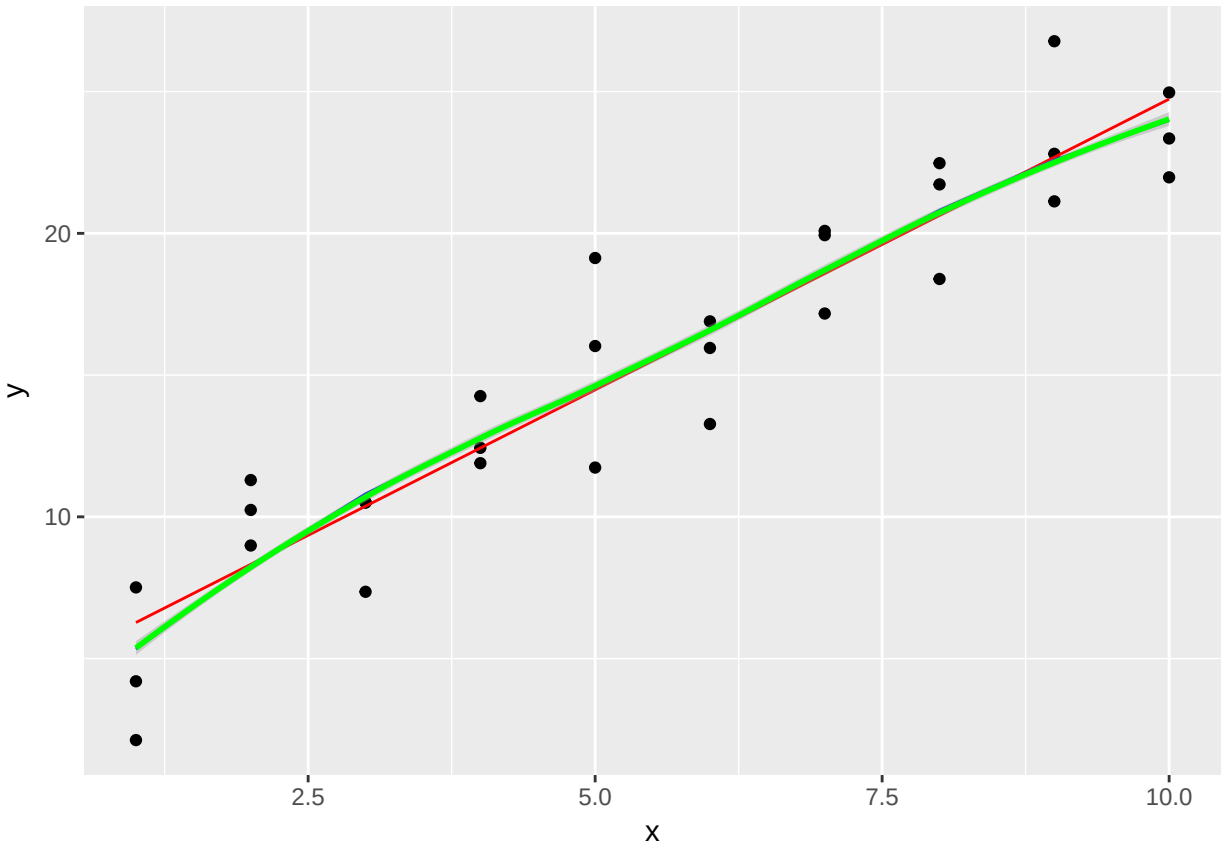
```
sim1_mod <- lm(y ~ x, data = sim1)
sim1_loess <- loess(y ~ x, data = sim1, degree = 2)

grid <- sim1 %>% data_grid(x)

grid_mod <- grid %>% add_predictions(sim1_mod)
grid_loess <- grid %>% add_predictions(sim1_loess)

ggplot(sim1, aes(x = x)) + geom_point(aes(y = y)) +
  geom_line(data = grid_mod, aes(y = pred), color = 'red') +
  geom_line(data = grid_loess, aes(y = pred), color = 'blue') +
  geom_smooth(data = grid_loess, aes(y = pred), color = 'green')
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



The line using geom_smooth (green) completely covers the line (blue) that went through the given process, so they're the same line (the red line is just a regular line from a linear model).

#3 What does geom_ref_line() do? What package does it come from? Why is displaying a reference line in plots showing residuals useful and important?

The function adds a horizontal or vertical reference line. It's from ggplot2. Displaying a reference is useful because it makes it easier to see if the residuals are randomly spread out around the line y = 0.

#4 Why might you want to look at a frequency polygon of absolute residuals? What are the pros and cons compared to looking at the raw residuals

A frquency polygon will help you calibrate the quality of the model, but it won't give you information about whether the residuals are randomly spread out as x increases or decreases.

23.4.5

#1 What happens if you repeat the analysis of sim2 using a model without an intercept. What happens to the model equation? What happens to the predictions?

```r
mod_i <- lm(y ~ x, data = sim2)
mod_ni <- lm(y ~ 0 + x, data = sim2)
pander(mod_i)
```

Table 2: Fitting linear model: y ~ x

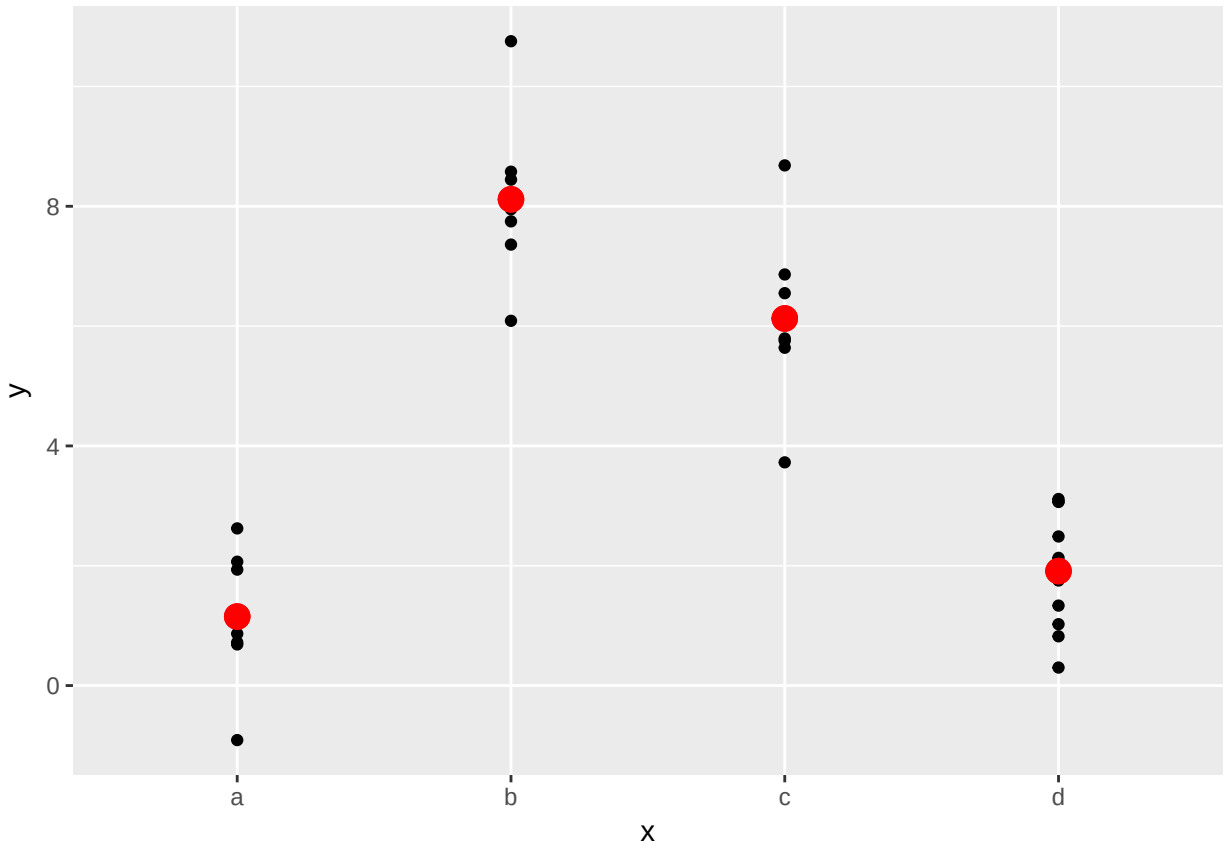
	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.152	0.3475	3.316	0.002095
xb	6.964	0.4914	14.17	2.684e-16
xc	4.975	0.4914	10.12	4.469e-12
xd	0.7588	0.4914	1.544	0.1313

```r
pander(mod_ni)
```

Table 3: Fitting linear model: y ~ 0 + x

	Estimate	Std. Error	t value	Pr(>|t|)
xa	1.152	0.3475	3.316	0.002095
xb	8.116	0.3475	23.36	2.428e-23
xc	6.127	0.3475	17.63	2.709e-19
xd	1.911	0.3475	5.499	3.245e-06

```r
grid <- sim2 %>%
  data_grid(x) %>%
  gather_predictions(mod_i, mod_ni)
ggplot(sim2, aes(x)) +
  geom_point(aes(y = y)) +
  geom_point(data = grid, aes(y = pred), colour = "red", size = 4)
```



If we repeat the analysis of sim2 using a model without an intercept, the equation changes, but the predictions don't change.

#3 Using the basic principles, convert the formulas in the following two models into functions. (Hint: start by converting the categorical variable into 0-1 variables.)

```
fun1 <- function(data, x, y, z){
  a <- x
  b <- y
  c <- z
  lm(a ~ b + c, data = data)
}
pander(fun1(sim3, sim3$y, sim3$x1, sim3$x2))
```

Table 4: Fitting linear model: a ~ b + c

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.872	0.3874	4.832	4.218e-06
b	-0.1967	0.04871	-4.039	9.718e-05
cb	2.888	0.3957	7.298	4.066e-11
cc	4.806	0.3957	12.14	2.42e-22
cd	2.36	0.3957	5.963	2.787e-08

```
pander(lm(y ~ x1 + x2, data = sim3))
```

Table 5: Fitting linear model: y ~ x1 + x2

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.872	0.3874	4.832	4.218e-06
x1	-0.1967	0.04871	-4.039	9.718e-05
x2b	2.888	0.3957	7.298	4.066e-11
x2c	4.806	0.3957	12.14	2.42e-22
x2d	2.36	0.3957	5.963	2.787e-08

```
fun2 <- function(data, x, y, z){
  a <- x
  b <- y
  c <- z
  lm(a ~ b * c, data = data)
}
pander(fun2(sim3, sim3$y, sim3$x1, sim3$x2))
```

Table 6: Fitting linear model: a ~ b * c

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.301	0.404	3.221	0.001673
b	-0.09302	0.06511	-1.429	0.1559
cb	7.069	0.5713	12.37	1.092e-22
cc	4.431	0.5713	7.755	4.407e-12
cd	0.8346	0.5713	1.461	0.1469
b:cb	-0.7603	0.09208	-8.257	3.301e-13
b:cc	0.06815	0.09208	0.7402	0.4608
b:cd	0.2773	0.09208	3.011	0.003216

```
pander(lm(y ~ x1 * x2, data = sim3))
```

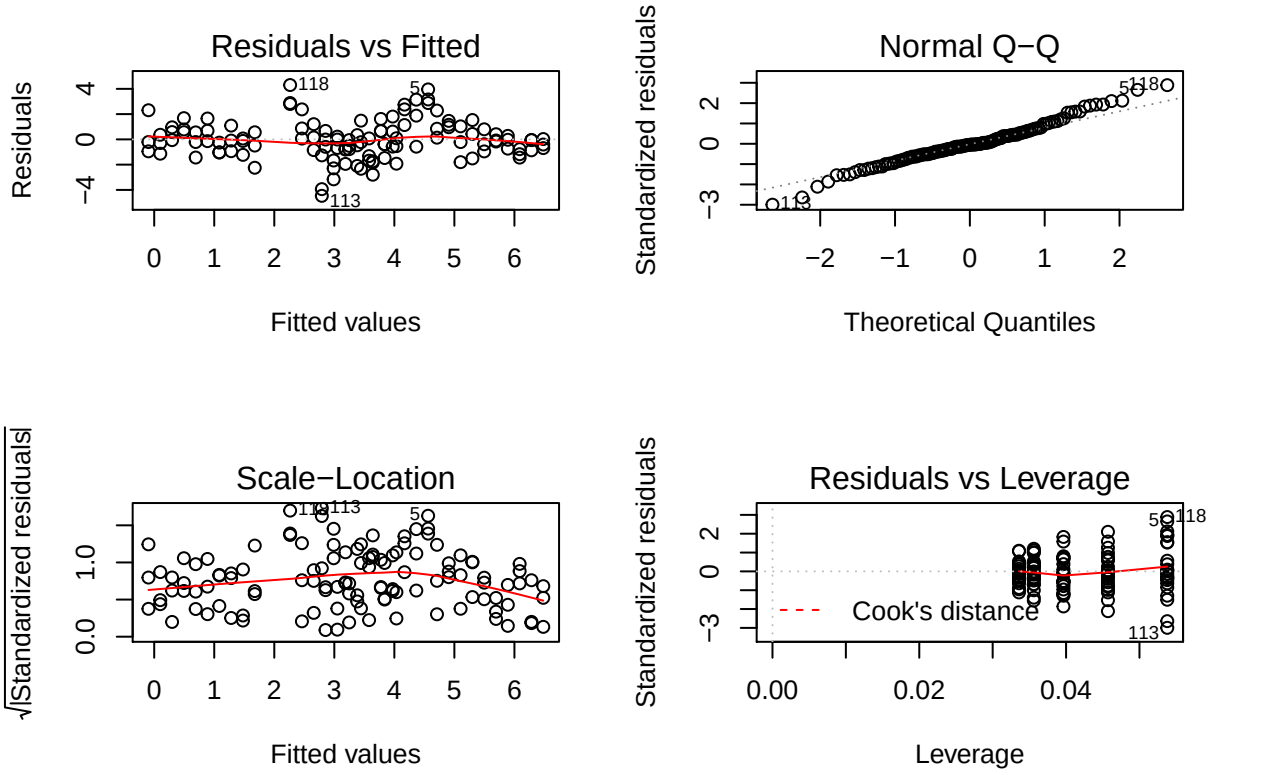
Table 7: Fitting linear model: y ~ x1 * x2

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.301	0.404	3.221	0.001673
x1	-0.09302	0.06511	-1.429	0.1559
x2b	7.069	0.5713	12.37	1.092e-22
x2c	4.431	0.5713	7.755	4.407e-12
x2d	0.8346	0.5713	1.461	0.1469
x1:x2b	-0.7603	0.09208	-8.257	3.301e-13
x1:x2c	0.06815	0.09208	0.7402	0.4608
x1:x2d	0.2773	0.09208	3.011	0.003216

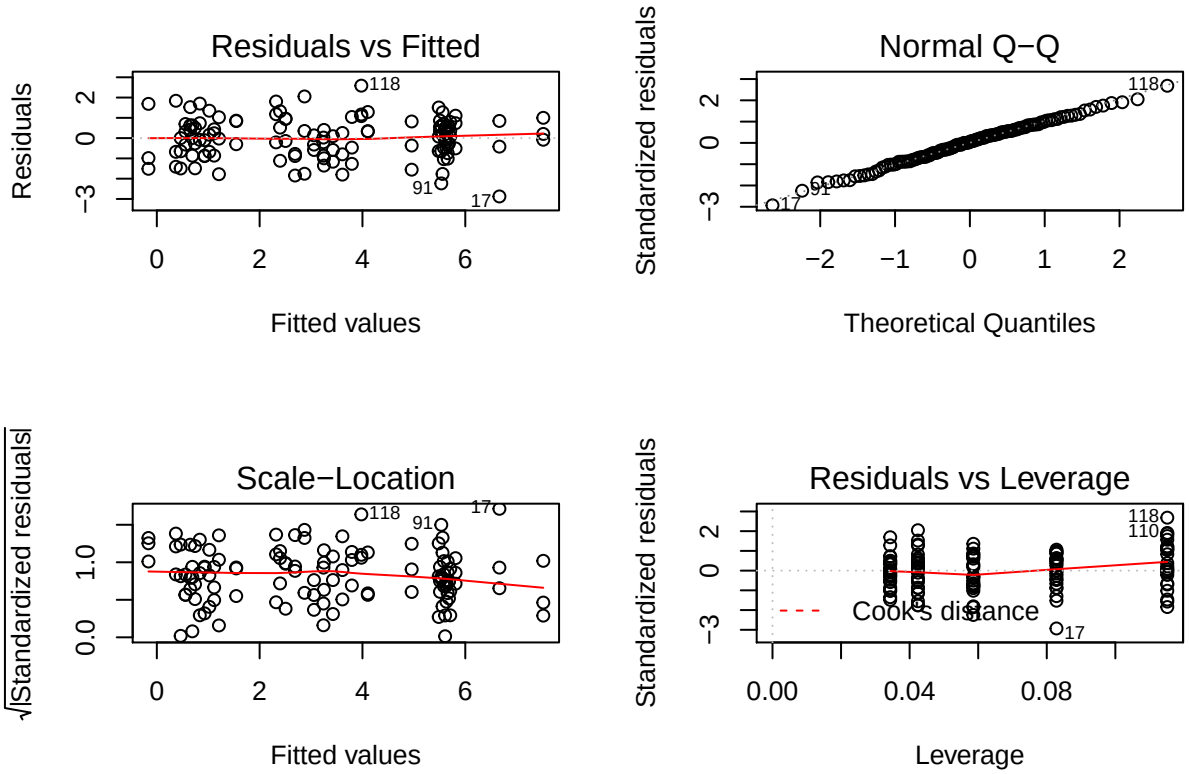
I'm not really sure if this is what the question was asking; it requires more typing than just using "lm" and the output is confusing.

#4 For sim4, which of mod1 and mod2 is better? I think mod2 does a slightly better job at removing patterns, but it's pretty subtle. Can you come up with a plot to support my claim?

```
mod1 <- lm(y ~ x1 + x2, data = sim3)
mod2 <- lm(y ~ x1 * x2, data = sim3)
par(mfrow = c(2,2))
plot(mod1)
```




```
par(mfrow = c(2,2))
plot(mod2)
```



The second model is better because the residuals of the second model follow the line better in the Q-Q plot and the residuals are randomly spread out (as seen in the Residuals vs Fitted plot), while the residuals in the first model start to deviate slighlty from the line as the theoretical quantities increase and there is a zig-zagging pattern in the Residuals vs Fitted plot.