Homework 4

# 23.2.1 Exercises

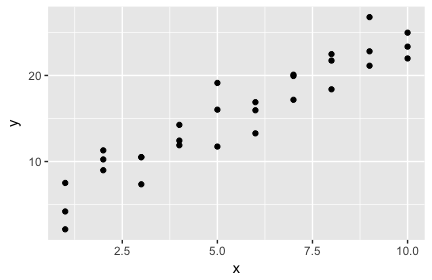
## (1) 23.2.1 Exercise 1

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 everything 6 times to generate different simulated datasets, and also examine the slope and intercept of the generated models, as well as the R2R2 values. How much variability do you see? How do the data values affect the model?

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

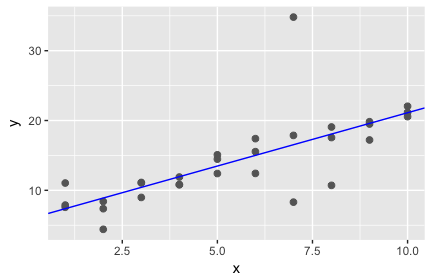
Hint: To make it easy to do this 6 times, it makes sense to wrap everything into a function, and call that from within a for loop. The following code will help you accomplish this:

p <- ggplot(sim1, aes(x, y)) +   
 geom\_point()  
print(p)

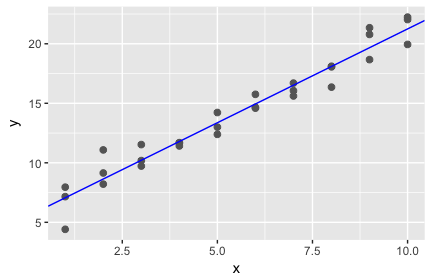


#Answer in a for loop  
for(i in 1:6){  
 sim1a <- tibble(  
 x=rep(1:10, each=3),  
 y=x\*1.5+6+rt(length(x), df=2)  
 )  
  
 sim1a\_mod <- lm(y ~x, data = sim1a)  
 coef(sim1a\_mod) %>% str()  
 p3 <- summary(sim1a\_mod)$r.squared \* 100  
 print(p3)  
   
 p4 <- ggplot(sim1a, aes(x, y)) +  
 geom\_point(size = 2, color = "grey40") +  
 geom\_abline(intercept = coef(sim1a\_mod)[1], slope = coef(sim1a\_mod)[2], color = "blue")  
 print(p4)  
   
 grid <- sim1a %>%  
 add\_predictions(sim1a\_mod)  
 print(grid)  
}

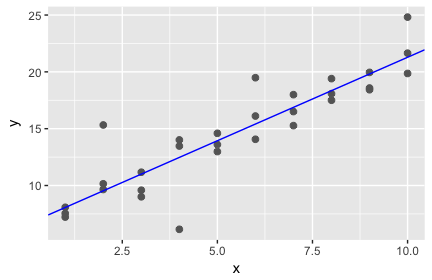
## Named num [1:2] 5.84 1.53  
## - attr(\*, "names")= chr [1:2] "(Intercept)" "x"  
## [1] 52.38643



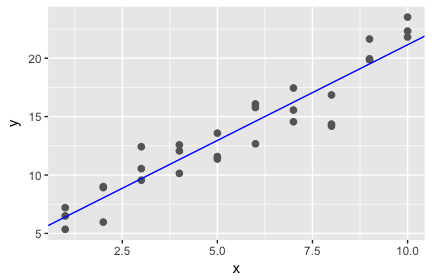
## # A tibble: 30 × 3  
## x y pred  
## <int> <dbl> <dbl>  
## 1 1 7.885250 7.368506  
## 2 1 7.574180 7.368506  
## 3 1 11.044440 7.368506  
## 4 2 7.367130 8.896560  
## 5 2 4.415072 8.896560  
## 6 2 8.405972 8.896560  
## 7 3 11.126571 10.424615  
## 8 3 8.981615 10.424615  
## 9 3 11.058926 10.424615  
## 10 4 11.904767 11.952669  
## # ... with 20 more rows  
## Named num [1:2] 5.48 1.58  
## - attr(\*, "names")= chr [1:2] "(Intercept)" "x"  
## [1] 95.1288



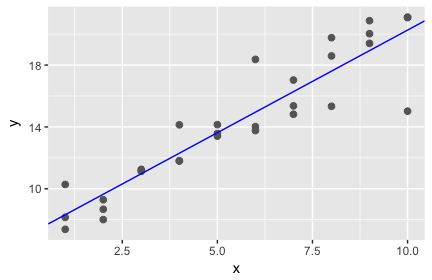
## # A tibble: 30 × 3  
## x y pred  
## <int> <dbl> <dbl>  
## 1 1 7.953289 7.061707  
## 2 1 7.157980 7.061707  
## 3 1 4.413136 7.061707  
## 4 2 9.144851 8.638458  
## 5 2 8.208490 8.638458  
## 6 2 11.083430 8.638458  
## 7 3 9.725887 10.215210  
## 8 3 11.531204 10.215210  
## 9 3 10.185942 10.215210  
## 10 4 11.694018 11.791961  
## # ... with 20 more rows  
## Named num [1:2] 6.6 1.47  
## - attr(\*, "names")= chr [1:2] "(Intercept)" "x"  
## [1] 80.58101



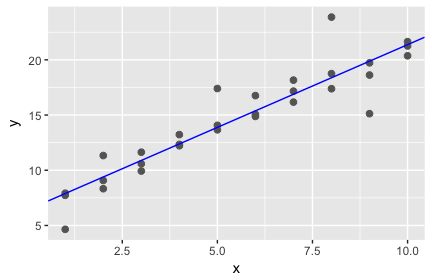
## # A tibble: 30 × 3  
## x y pred  
## <int> <dbl> <dbl>  
## 1 1 8.083290 8.065493  
## 2 1 7.544008 8.065493  
## 3 1 7.220822 8.065493  
## 4 2 9.650425 9.534419  
## 5 2 10.151616 9.534419  
## 6 2 15.326875 9.534419  
## 7 3 9.006018 11.003345  
## 8 3 11.169978 11.003345  
## 9 3 9.586089 11.003345  
## 10 4 6.145894 12.472270  
## # ... with 20 more rows  
## Named num [1:2] 4.77 1.64  
## - attr(\*, "names")= chr [1:2] "(Intercept)" "x"  
## [1] 89.77092



## # A tibble: 30 × 3  
## x y pred  
## <int> <dbl> <dbl>  
## 1 1 6.496142 6.407972  
## 2 1 7.202975 6.407972  
## 3 1 5.353572 6.407972  
## 4 2 9.012469 8.045939  
## 5 2 5.968226 8.045939  
## 6 2 8.916935 8.045939  
## 7 3 9.556528 9.683906  
## 8 3 12.420495 9.683906  
## 9 3 10.544939 9.683906  
## 10 4 12.583750 11.321873  
## # ... with 20 more rows  
## Named num [1:2] 6.99 1.33  
## - attr(\*, "names")= chr [1:2] "(Intercept)" "x"  
## [1] 85.29437



## # A tibble: 30 × 3  
## x y pred  
## <int> <dbl> <dbl>  
## 1 1 7.379553 8.320191  
## 2 1 10.278083 8.320191  
## 3 1 8.153150 8.320191  
## 4 2 8.673660 9.647073  
## 5 2 8.007227 9.647073  
## 6 2 9.289446 9.647073  
## 7 3 11.254763 10.973956  
## 8 3 11.135052 10.973956  
## 9 3 11.167043 10.973956  
## 10 4 11.796950 12.300839  
## # ... with 20 more rows  
## Named num [1:2] 6.39 1.5  
## - attr(\*, "names")= chr [1:2] "(Intercept)" "x"  
## [1] 85.89221



## # A tibble: 30 × 3  
## x y pred  
## <int> <dbl> <dbl>  
## 1 1 4.653483 7.892027  
## 2 1 7.732199 7.892027  
## 3 1 7.920110 7.892027  
## 4 2 9.076493 9.390524  
## 5 2 11.331285 9.390524  
## 6 2 8.329920 9.390524  
## 7 3 11.634677 10.889021  
## 8 3 9.928534 10.889021  
## 9 3 10.588533 10.889021  
## 10 4 12.341385 12.387518  
## # ... with 20 more rows

There is some variability, but only to a certain extent. Slopes seem to be contained between the following (1< slope < 2), and the intercept varies between (4 < intercept < 7). This is for the observance of the data, but if we were to generate more datasets and especially if we were to change the method in which we are doing so, we would see different slopes and intercepts. The data shifts the data up and down and the slope slightly, but overall the data seems to all have a positive slope within the above parameters and intercepts.

--Still need to write how the data affects the models

## (2) 23.2.1 Exercise 2

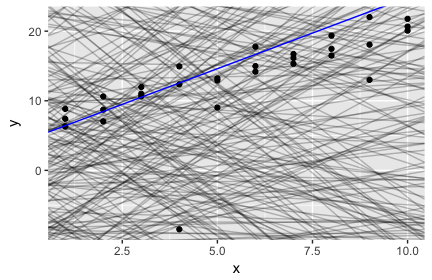
One way to make linear models more robust is to use a different distance measure. For example, instead of root-mean-squared (RMS) distance, you could use mean-absolute-deviation (MAD) distance:

Use optim() to fit this model to the simulated data above and visually compare it to the linear model, again examining at least 6 instances. You want both models to be drawn on top of each instance of the simulated data set. Discuss what you find.

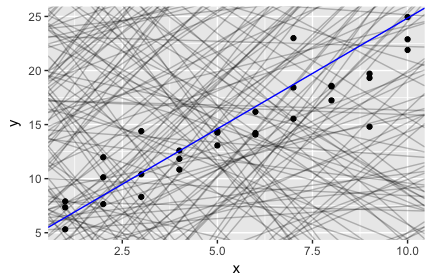
Hints: make\_prediction() is not a built-in function; you will need to create your own linear function. Unlike Exercise 1, here you should only compare the models graphically; I don’t think the R2R2 metric is meaningful for a model that uses the MAD distance measure.

#Answer in a for loop  
  
make\_prediction <- function(a, data) {  
 a[1] + data$x\*a[2]  
}  
  
measure\_distance <- function(a, data) {  
 diff <- data$y - make\_prediction(a, data)  
 mean(abs(diff))  
}  
  
for(i in 1:6){  
 sim1a <- tibble(  
 x=rep(1:10, each=3),  
 y=x\*1.5+6+rt(length(x), df=2)  
 )  
  
 models <- tibble(  
 a0 = runif(250, -20, 40),  
 a1 = runif(250, -5, 5)  
 )  
   
 best <- optim(c(0, 0), measure\_distance, data = sim1)  
 best %>% str()  
  
 p2 <- ggplot(sim1a, aes(x,y)) + geom\_abline(aes(intercept = a0, slope=a1), data = models, alpha = 1/4) + geom\_point() + geom\_abline(intercept = best$par[1], slope = best$par[2], color = "blue")  
 print(p2)  
   
   
}

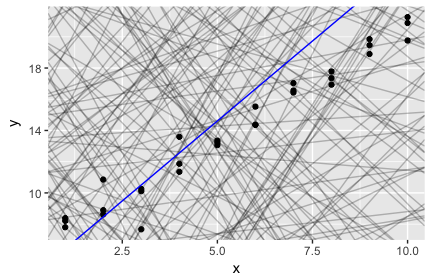
## List of 5  
## $ par : num [1:2] 4.36 2.05  
## $ value : num 1.7  
## $ counts : Named int [1:2] 119 NA  
## ..- attr(\*, "names")= chr [1:2] "function" "gradient"  
## $ convergence: int 0  
## $ message : NULL



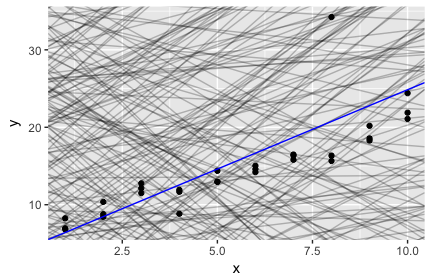
## List of 5  
## $ par : num [1:2] 4.36 2.05  
## $ value : num 1.7  
## $ counts : Named int [1:2] 119 NA  
## ..- attr(\*, "names")= chr [1:2] "function" "gradient"  
## $ convergence: int 0  
## $ message : NULL



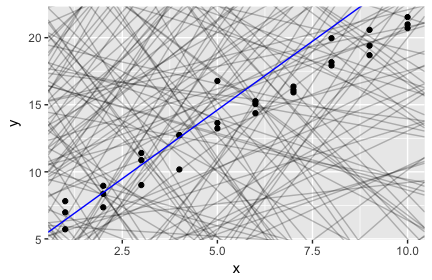
## List of 5  
## $ par : num [1:2] 4.36 2.05  
## $ value : num 1.7  
## $ counts : Named int [1:2] 119 NA  
## ..- attr(\*, "names")= chr [1:2] "function" "gradient"  
## $ convergence: int 0  
## $ message : NULL



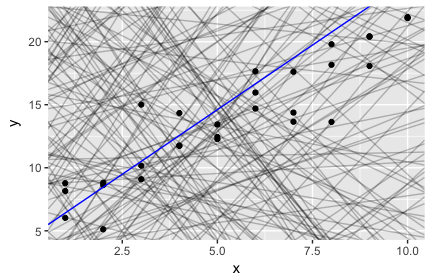
## List of 5  
## $ par : num [1:2] 4.36 2.05  
## $ value : num 1.7  
## $ counts : Named int [1:2] 119 NA  
## ..- attr(\*, "names")= chr [1:2] "function" "gradient"  
## $ convergence: int 0  
## $ message : NULL



## List of 5  
## $ par : num [1:2] 4.36 2.05  
## $ value : num 1.7  
## $ counts : Named int [1:2] 119 NA  
## ..- attr(\*, "names")= chr [1:2] "function" "gradient"  
## $ convergence: int 0  
## $ message : NULL



## List of 5  
## $ par : num [1:2] 4.36 2.05  
## $ value : num 1.7  
## $ counts : Named int [1:2] 119 NA  
## ..- attr(\*, "names")= chr [1:2] "function" "gradient"  
## $ convergence: int 0  
## $ message : NULL



# 23.3.3 Exercises

## (3) 23.3.3 Exercise 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()?

sim1a <- tibble(  
 x=rep(1:10, each=3),  
 y=x\*1.5+6+rt(length(x), df=2)  
 )  
  
 sim1a\_mod <- loess(y ~x, data = sim1a)  
 coef(sim1a\_mod) %>% str()

## NULL

p3 <- summary(sim1a\_mod)$r.squared \* 100  
 print(p3)

## numeric(0)

sim1a\_mod2 <- lm(y ~x, data = sim1a)  
 coef(sim1a\_mod2) %>% str()

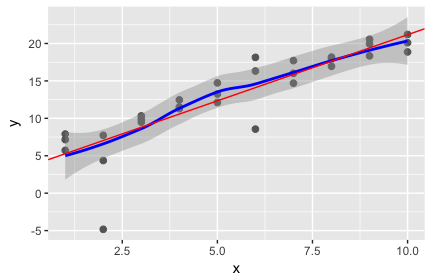
## Named num [1:2] 3.5 1.77  
## - attr(\*, "names")= chr [1:2] "(Intercept)" "x"

p3b <- summary(sim1a\_mod2)$r.squared  
 print(p3b)

## [1] 0.7620095

p4 <- ggplot(sim1a, aes(x, y)) +  
 geom\_point(size = 2, color = "grey40") +  
 geom\_smooth(color = "blue") +  
 geom\_abline(intercept = coef(sim1a\_mod2)[1], slope = coef(sim1a\_mod2)[2], color = "red")  
 print(p4)

## `geom\_smooth()` using method = 'loess'



grid <- sim1a %>%  
 add\_predictions(sim1a\_mod)  
 print(grid)

## # A tibble: 30 × 3  
## x y pred  
## <int> <dbl> <dbl>  
## 1 1 7.157071 5.007020  
## 2 1 7.899507 5.007020  
## 3 1 5.725148 5.007020  
## 4 2 4.362244 6.590022  
## 5 2 7.729406 6.590022  
## 6 2 -4.827568 6.590022  
## 7 3 10.326389 8.611582  
## 8 3 9.484209 8.611582  
## 9 3 9.901263 8.611582  
## 10 4 11.518147 11.327328  
## # ... with 20 more rows

## (4) 23.3.3 Exercise 2

add\_predictions() is paired with gather\_predictions() and spread\_predictions(). Use both gather\_predictions() and spread\_predictions() to repeat the analysis with the two models used in problem (3) above. How do these three functions differ?

sim1a

## # A tibble: 30 × 2  
## x y  
## <int> <dbl>  
## 1 1 7.157071  
## 2 1 7.899507  
## 3 1 5.725148  
## 4 2 4.362244  
## 5 2 7.729406  
## 6 2 -4.827568  
## 7 3 10.326389  
## 8 3 9.484209  
## 9 3 9.901263  
## 10 4 11.518147  
## # ... with 20 more rows

grid <- sim1a %>%  
 add\_predictions(sim1a\_mod2) %>%  
 add\_predictions(sim1a\_mod) %>%  
print(grid)

## # A tibble: 30 × 3  
## x y pred  
## <int> <dbl> <dbl>  
## 1 1 7.157071 5.007020  
## 2 1 7.899507 5.007020  
## 3 1 5.725148 5.007020  
## 4 2 4.362244 6.590022  
## 5 2 7.729406 6.590022  
## 6 2 -4.827568 6.590022  
## 7 3 10.326389 8.611582  
## 8 3 9.484209 8.611582  
## 9 3 9.901263 8.611582  
## 10 4 11.518147 11.327328  
## # ... with 20 more rows

grid <- sim1a %>%  
 gather\_predictions(sim1a\_mod2, sim1a\_mod) %>%  
print(grid1)

## # A tibble: 60 × 4  
## model x y pred  
## <chr> <int> <dbl> <dbl>  
## 1 sim1a\_mod2 1 7.157071 5.268896  
## 2 sim1a\_mod2 1 7.899507 5.268896  
## 3 sim1a\_mod2 1 5.725148 5.268896  
## 4 sim1a\_mod2 2 4.362244 7.036693  
## 5 sim1a\_mod2 2 7.729406 7.036693  
## 6 sim1a\_mod2 2 -4.827568 7.036693  
## 7 sim1a\_mod2 3 10.326389 8.804490  
## 8 sim1a\_mod2 3 9.484209 8.804490  
## 9 sim1a\_mod2 3 9.901263 8.804490  
## 10 sim1a\_mod2 4 11.518147 10.572287  
## # ... with 50 more rows

grid <- sim1a %>%  
 spread\_predictions(sim1a\_mod2, sim1a\_mod) %>%  
print(grid2)

## # A tibble: 30 × 4  
## x y sim1a\_mod2 sim1a\_mod  
## <int> <dbl> <dbl> <dbl>  
## 1 1 7.157071 5.268896 5.007020  
## 2 1 7.899507 5.268896 5.007020  
## 3 1 5.725148 5.268896 5.007020  
## 4 2 4.362244 7.036693 6.590022  
## 5 2 7.729406 7.036693 6.590022  
## 6 2 -4.827568 7.036693 6.590022  
## 7 3 10.326389 8.804490 8.611582  
## 8 3 9.484209 8.804490 8.611582  
## 9 3 9.901263 8.804490 8.611582  
## 10 4 11.518147 10.572287 11.327328  
## # ... with 20 more rows

Add\_predictions adds a single column with predictions. Gather\_predictions adds a column for models and a column for predictions, and spread\_predictions adds a column for each models' predictions.

## (5) 23.3.3 Exercise 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?

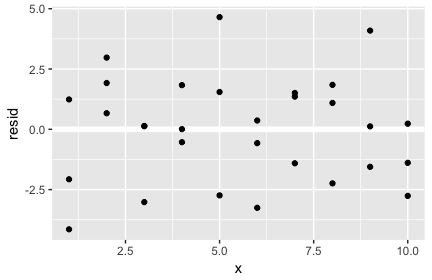
sim1\_mod <- lm(y ~ x, data = sim1)  
coef(sim1\_mod)

## (Intercept) x   
## 4.220822 2.051533

sim1 <- sim1 %>%   
 add\_residuals(sim1\_mod)  
sim1

## # A tibble: 30 × 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

ggplot(sim1, aes(x, resid)) +   
 geom\_ref\_line(h = 0) +  
 geom\_point()



It adds a reference line for comparison. It is contained in the modelr package. It can give us a line in which to compare our residual values. This can make it more clear how the data is distributed. It can also make it more clear how a best fit line is oriented and why. If it looks randomly distributed then the model is fit well to the data.

## (6) 23.3.3 Exercise 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?

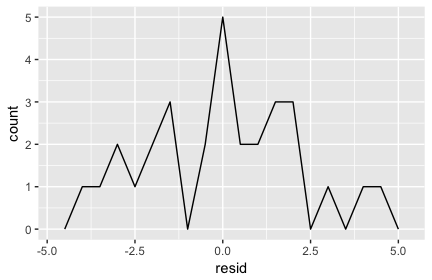
sim1\_mod <- lm(y ~ x, data = sim1)  
coef(sim1\_mod)

## (Intercept) x   
## 4.220822 2.051533

sim1 <- sim1 %>%   
 add\_residuals(sim1\_mod)  
sim1

## # A tibble: 30 × 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

ggplot(sim1, aes(resid)) +   
 geom\_freqpoly(binwidth = 0.5)



A frequency polygon of absolute residuals can supply us with data about the distribution of residual values within our data and a fitted model. Looking at the raw residuals can give us specific values and probably a better idea of orientation of these based upon differences, where as a frequency polygon depending upon the binwidth will give us an idea of how the residuals are distributed from a bigger picture standpoint.

# 23.4.5 Exercises

## (7) 23.4.5 Exercise 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?

sim2

## # A tibble: 40 × 2  
## x y  
## <chr> <dbl>  
## 1 a 1.9353632  
## 2 a 1.1764886  
## 3 a 1.2436855  
## 4 a 2.6235489  
## 5 a 1.1120381  
## 6 a 0.8660030  
## 7 a -0.9100875  
## 8 a 0.7207628  
## 9 a 0.6865540  
## 10 a 2.0673079  
## # ... with 30 more rows

sim2\_mod <- lm(y ~0, data = sim2)  
 coef(sim2\_mod) %>% str()

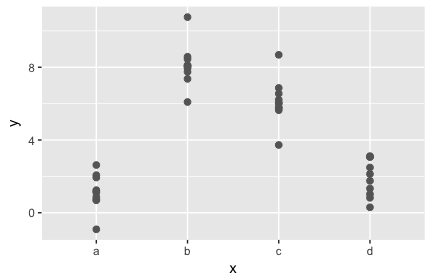
## num(0)

p3 <- summary(sim2\_mod)$r.squared \* 100  
 print(p3)

## [1] 0

p4 <- ggplot(sim2, aes(x, y)) +  
 geom\_point(size = 2, color = "grey40") +  
 geom\_abline(slope = coef(sim2\_mod)[1], color = "blue")  
 print(p4)

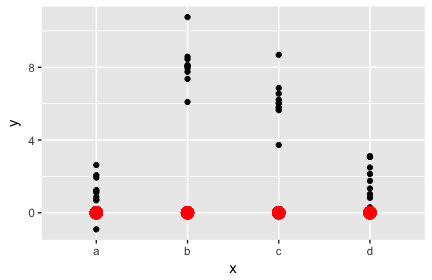
## Warning: Removed 1 rows containing missing values (geom\_abline).



grid <- sim2 %>%  
 add\_predictions(sim2\_mod) %>%  
 print(grid)

## # A tibble: 40 × 3  
## x y pred  
## <chr> <dbl> <dbl>  
## 1 a 1.9353632 0  
## 2 a 1.1764886 0  
## 3 a 1.2436855 0  
## 4 a 2.6235489 0  
## 5 a 1.1120381 0  
## 6 a 0.8660030 0  
## 7 a -0.9100875 0  
## 8 a 0.7207628 0  
## 9 a 0.6865540 0  
## 10 a 2.0673079 0  
## # ... with 30 more rows

ggplot(sim2, aes(x)) + geom\_point(aes(y=y)) +   
 geom\_point(data = grid, aes(y=pred), colour="red", size=4)



The model equation is removed because there is a missing value of intercept for geom\_abline. The predictions are all at y-value 0 across the four x-values on the x-axis.

## (8) 23.4.5 Exercise 4

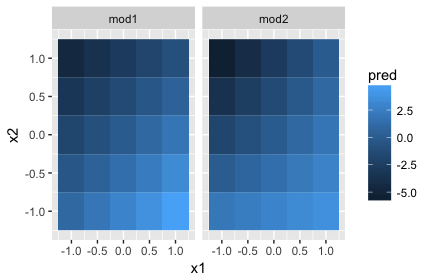
For sim4, which of mod1 and mod2 is better? The book author believes that mod2 does a slightly better job at removing patterns, but claims the effects are subtle. Examine the R2R2 values for the two models, and develop a plot that attempts to support this claim. In the end, how well does your plot support the claim? How likely is it that the claim is true?

Both seem fairly equally good for this simulation. I believe that mod1 is slightly more consistent and is thus a probably not as good of a model because we have learned that if it seems more random, then it is probably a better fit model. Mod 2, on the other hand does not have as consistent of a trend and thus carries slightly more randomness in it and fits ever so slightly better. This will be tough to see, but I will attempt this below.

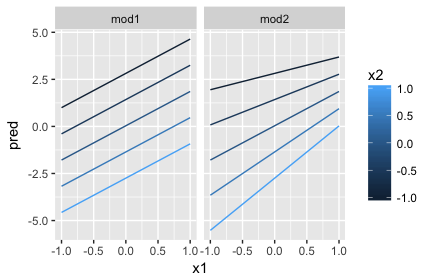
mod1 <- lm(y ~ x1 + x2, data = sim4)  
mod2 <- lm(y ~ x1 \* x2, data = sim4)  
  
grid <- sim4 %>%  
 add\_predictions(mod1) %>%  
 add\_predictions(mod2) %>%  
 data\_grid(  
 x1 = seq\_range(x1, 5),   
 x2 = seq\_range(x2, 5)   
 ) %>%   
 gather\_predictions(mod1, mod2)  
grid

## # A tibble: 50 × 4  
## model x1 x2 pred  
## <chr> <dbl> <dbl> <dbl>  
## 1 mod1 -1.0 -1.0 0.9963094  
## 2 mod1 -1.0 -0.5 -0.3949484  
## 3 mod1 -1.0 0.0 -1.7862061  
## 4 mod1 -1.0 0.5 -3.1774639  
## 5 mod1 -1.0 1.0 -4.5687216  
## 6 mod1 -0.5 -1.0 1.9071424  
## 7 mod1 -0.5 -0.5 0.5158847  
## 8 mod1 -0.5 0.0 -0.8753731  
## 9 mod1 -0.5 0.5 -2.2666308  
## 10 mod1 -0.5 1.0 -3.6578886  
## # ... with 40 more rows

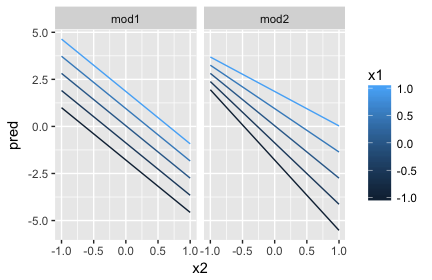
ggplot(grid, aes(x1, x2)) +   
 geom\_tile(aes(fill = pred)) +   
 facet\_wrap(~ model)



ggplot(grid, aes(x1, pred, colour = x2, group = x2)) +   
 geom\_line() +  
 facet\_wrap(~ model)



ggplot(grid, aes(x2, pred, colour = x1, group = x1)) +   
 geom\_line() +  
 facet\_wrap(~ model)



p3 <- summary(mod1)$r.squared \* 100  
 print(p3)

## [1] 50.5453

p3 <- summary(mod2)$r.squared \* 100  
 print(p3)

## [1] 52.2336