Lab 7 Solutions

Stats 32: Introduction to R for Undergraduates

Harrison Li

04/23/2024

Linear regression is probably the most fundamental statistical tool for prediction and modeling due to its simplicity and versatility. We will demonstrate it using the tidyverse's **mpg** dataset, which looks at the miles per gallon (mileage) of various cars, along with some of their other attributes:

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
summary(mpg)
```

```
##  manufacturer          model               displ            year     
##  Length:234         Length:234         Min.   :1.600   Min.   :1999  
##  Class :character   Class :character   1st Qu.:2.400   1st Qu.:1999  
##  Mode  :character   Mode  :character   Median :3.300   Median :2004  
##                                        Mean   :3.472   Mean   :2004  
##                                        3rd Qu.:4.600   3rd Qu.:2008  
##                                        Max.   :7.000   Max.   :2008  
##       cyl           trans               drv                 cty       
##  Min.   :4.000   Length:234         Length:234         Min.   : 9.00  
##  1st Qu.:4.000   Class :character   Class :character   1st Qu.:14.00  
##  Median :6.000   Mode  :character   Mode  :character   Median :17.00  
##  Mean   :5.889                                         Mean   :16.86  
##  3rd Qu.:8.000                                         3rd Qu.:19.00  
##  Max.   :8.000                                         Max.   :35.00  
##       hwy             fl               class          
##  Min.   :12.00   Length:234         Length:234        
##  1st Qu.:18.00   Class :character   Class :character  
##  Median :24.00   Mode  :character   Mode  :character  
##  Mean   :23.44                                        
##  3rd Qu.:27.00                                        
##  Max.   :44.00                                        
```

Exploratory analysis

Before jumping into any sort of statistical modeling, you always want to be clear about what scientific question you're trying to answer. For our purposes, let's say we're interested in understanding how to predict a car's highway mileage `hwy` from its city mileage `cty`.

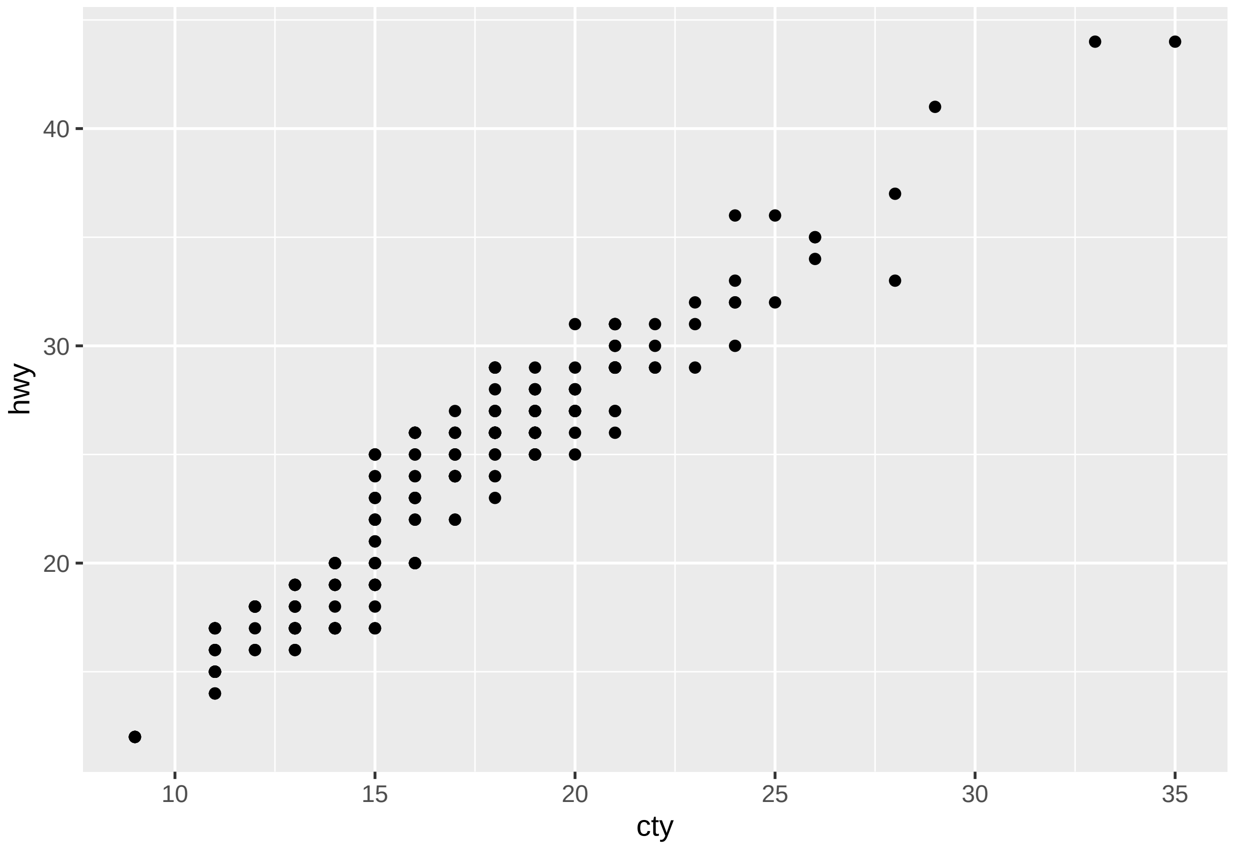
1. What are the dependent and independent variables in this setting?

Answer: `hwy` is the dependent variable (what we want to predict) and `cty` is the independent variable (what we use to predict the dependent variable).

Next, we do some exploratory data analysis to get an idea of your data's behavior. This often involves data wrangling and/or visualization, which is why we spent time in this class studying the `dplyr` and `ggplot2` packages!

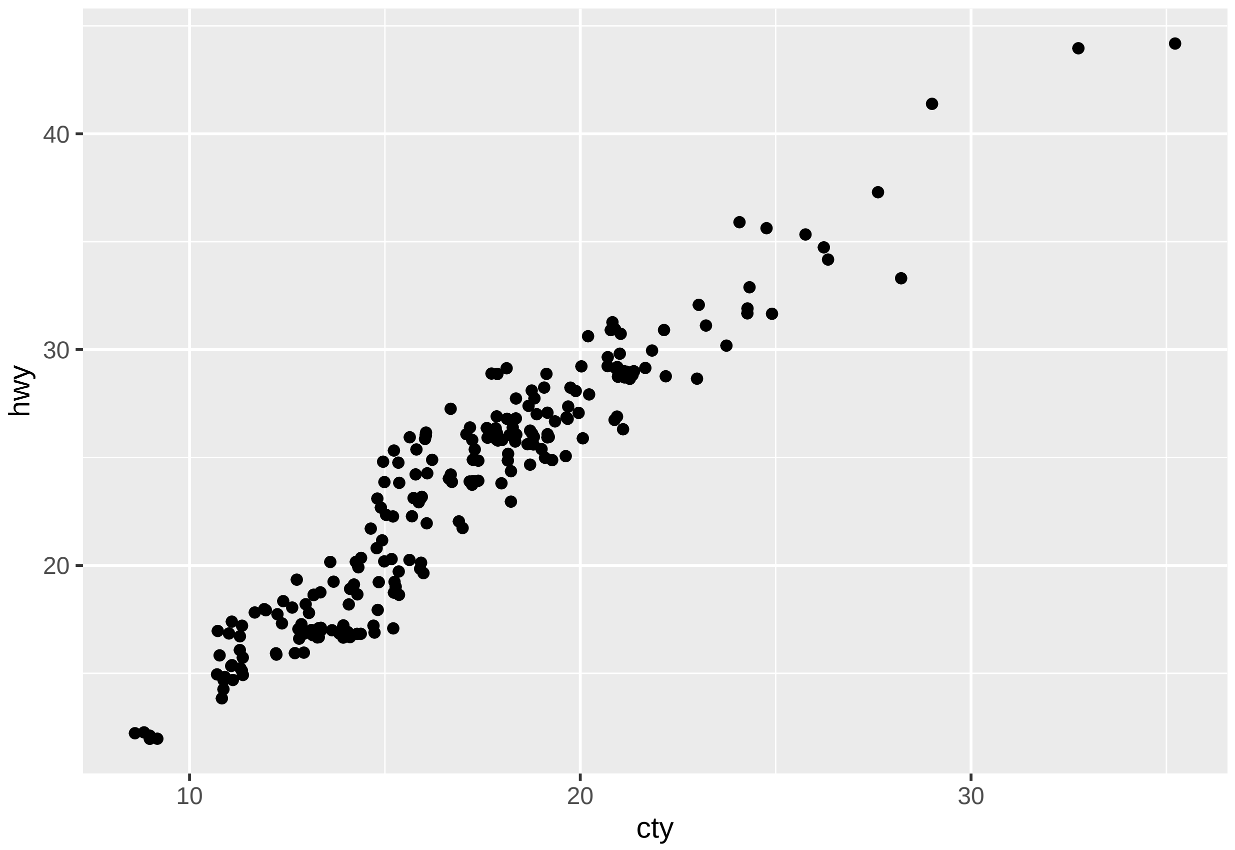
Since we have two quantitative variables, it's natural to look at a scatterplot:

```r
mpg %>%
  ggplot(aes(x=cty, y=hwy)) +
  geom_point()
```



To avoid having points right on top of each other (this is relevant here since we have a lot of cars, and their mileages are given as integers), we can use `geom_jitter()` instead of `geom_point()`

```r
mpg %>%
  ggplot(aes(x=cty, y=hwy)) +
  geom_jitter()
```



There indeed seems to be a strong, roughly linear relationship between y and x, justifying the use of a linear regression model.

To quantify the strength of the relationship, we can look at the correlation coefficient:

```
cor(mpg$cty, mpg$hwy)
```

```
## [1] 0.9559159
```

With a correlation close to +1, there is a strong positive relationship between city and highway mpg. This is consistent with our scatterplot.

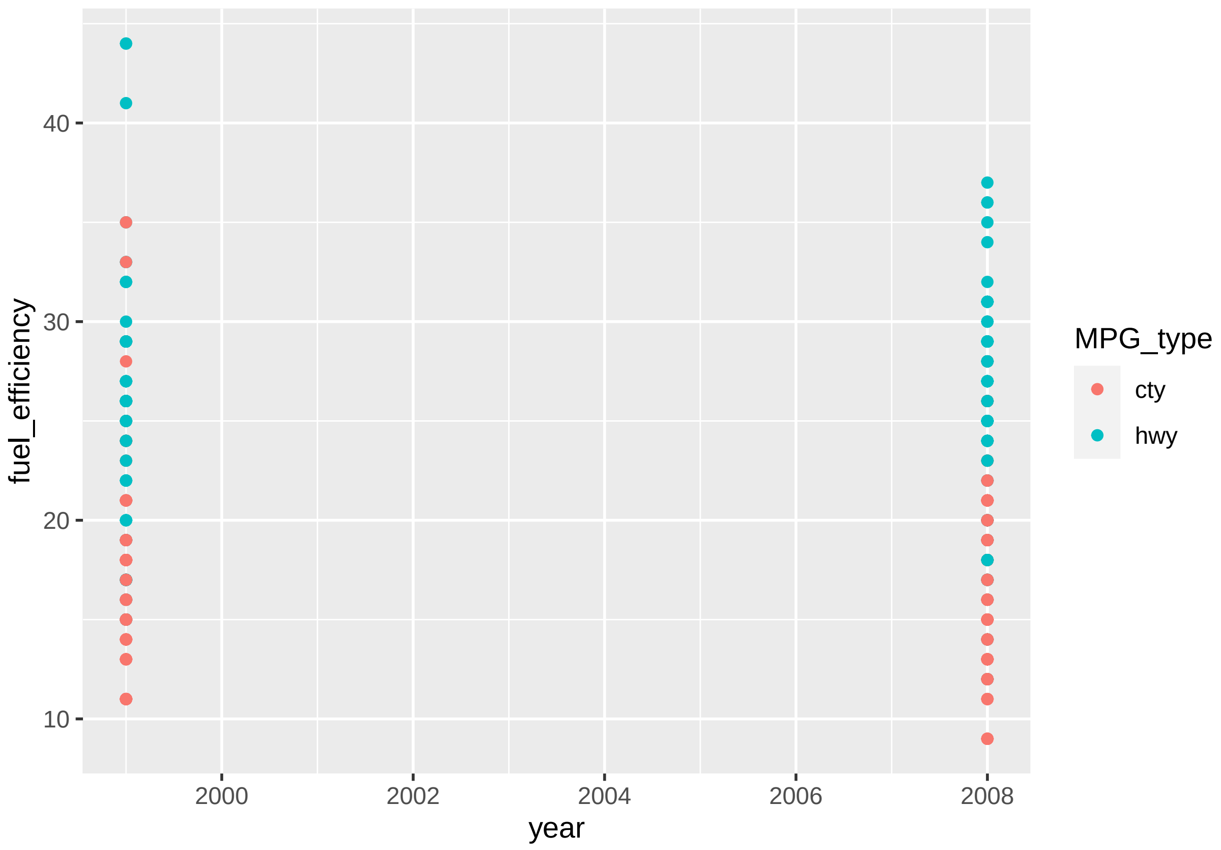
Note the correlation between x and y is the same as the correlation between y and x:

```
cor(mpg$hwy, mpg$cty) == cor(mpg$cty, mpg$hwy)
```

```
## [1] TRUE
```

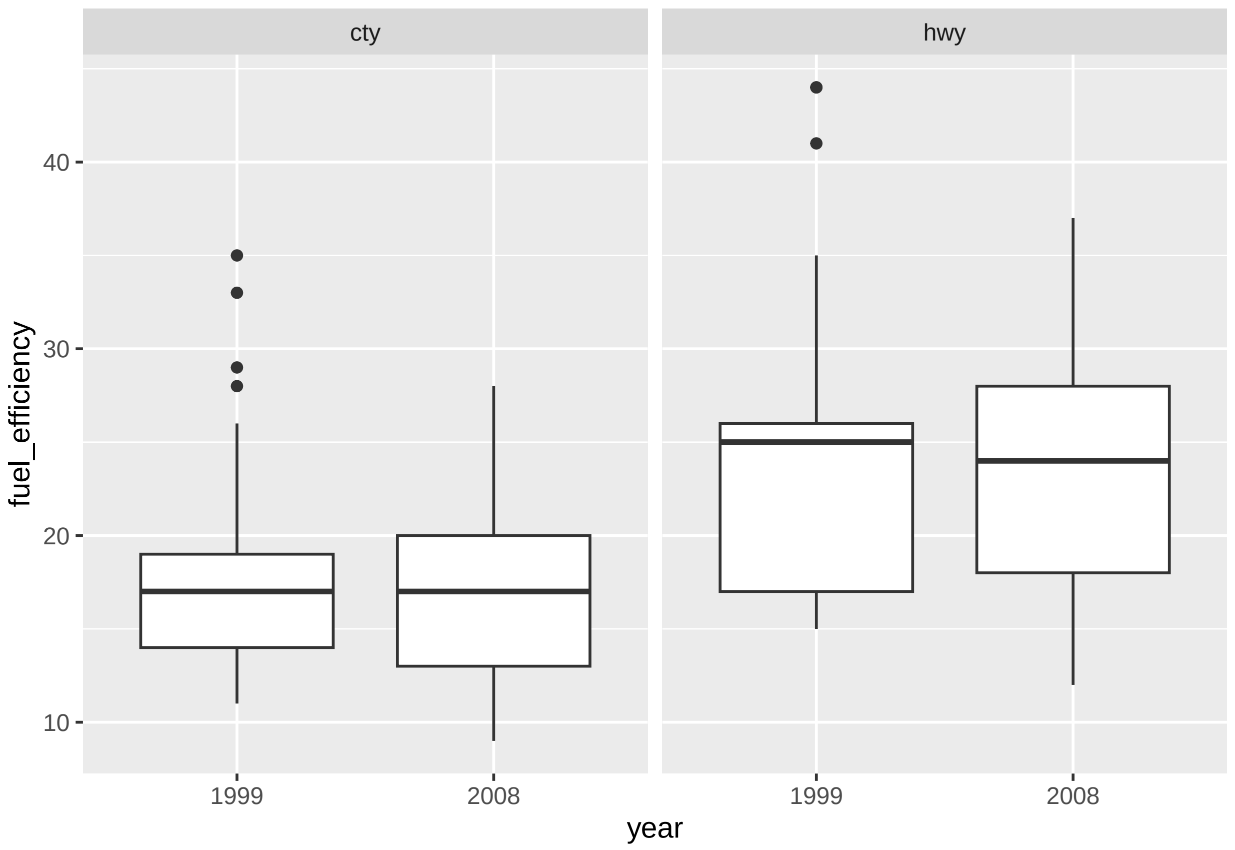
We now also visualize the relationship between fuel efficiency vs. year with a scatterplot. Note the use of `pivot_longer()` to create a longer version of the `mpg` tibble with twice the number of rows, containing a column called `MPG_type` that is either `cty` or `hwy`.

```
mpg %>%
  pivot_longer(names_to="MPG_type",
               values_to="fuel_efficiency",
               cols=c(cty, hwy)) %>%
  ggplot(aes(x=year, y=fuel_efficiency)) +
  geom_point(aes(colour=MPG_type))
```



Note that `year` apparently only takes on two possible values in this dataset: 1999 or 2008. This makes the scatterplot rather degenerate. So we prefer to treat `year` as categorical (indeed, binary) and create side-by-side boxplots with faceting:

```
mpg %>%
  mutate(year=as.factor(year)) %>%
  pivot_longer(names_to="MPG_type",
               values_to="fuel_efficiency",
               cols=c(cty, hwy)) %>%
  ggplot(aes(x=year, y=fuel_efficiency)) +
  geom_boxplot() +
  facet_wrap(~MPG_type)
```



Simple linear regression model

The linear trend seen in the scatterplot above justifies the use of a simple linear regression model:

$$y = \beta_0 + \beta_1 * x + e$$

Recall β_0 (the "intercept") and β_1 (the "slope") are unknown coefficients, and e represents random variation (noise).

We use the function `lm()` to fit the desired linear regression. The function outputs a large named list, consisting of a bunch of information pertinent to the model. Let's store this list in the variable `car_model`:

```
car_model <- lm(formula=hwy ~ cty, data=mpg)
```

Note the formula `hwy ~ cty`, which refers to the column names in the `mpg` tibble. `lm()` knows to look in the `mpg` tibble because we specified it in the `data` argument.

Let's extract the coefficient estimates from the model object `car_model`:

```
coefs <- car_model$coefficients
coefs
```

```
## (Intercept)         cty
##   0.8920411   1.3374556
```

Visualizing the regression line

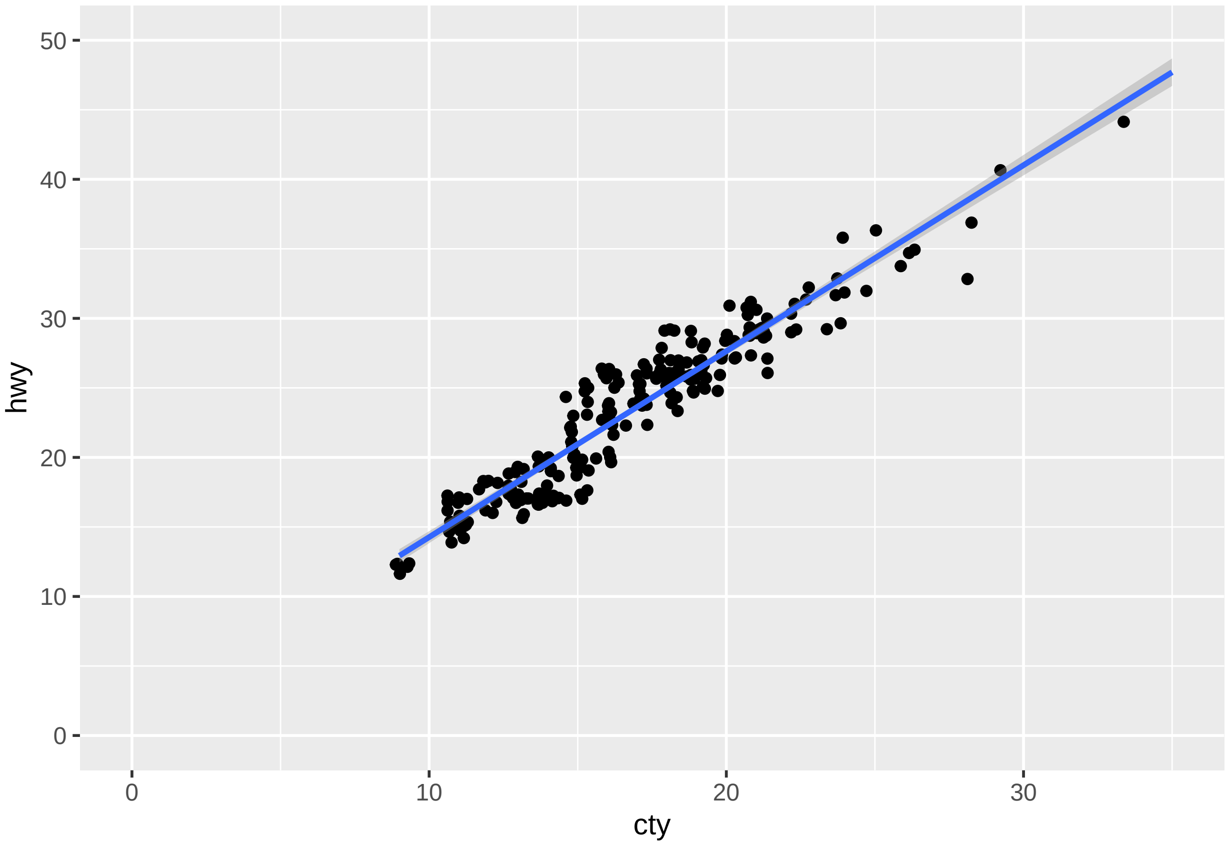
With the slope and intercept of the regression line, we can visualize it on top of the scatterplot above using `geom_abline()`, as in lecture.

However, this is such a popular visualization that `ggplot2` has an even easier way to do it, via `geom_smooth()`, that avoids you from explicitly needing to use `lm()` (the function calls `lm()` under the hood)

```
mpg %>%
  ggplot(aes(x=cty, y=hwy)) +
  geom_jitter() +
  geom_smooth(method="lm") +
  xlim(c(0, 35)) +
  ylim(c(0, 50))
```

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

```
## Warning: Removed 1 rows containing missing values (`geom_point()`).
```



Beautiful! The grey shading, visible near the top right of the plot, quantifies the uncertainty of the fit. If the line fit less well, you'd see a much wider grey region.

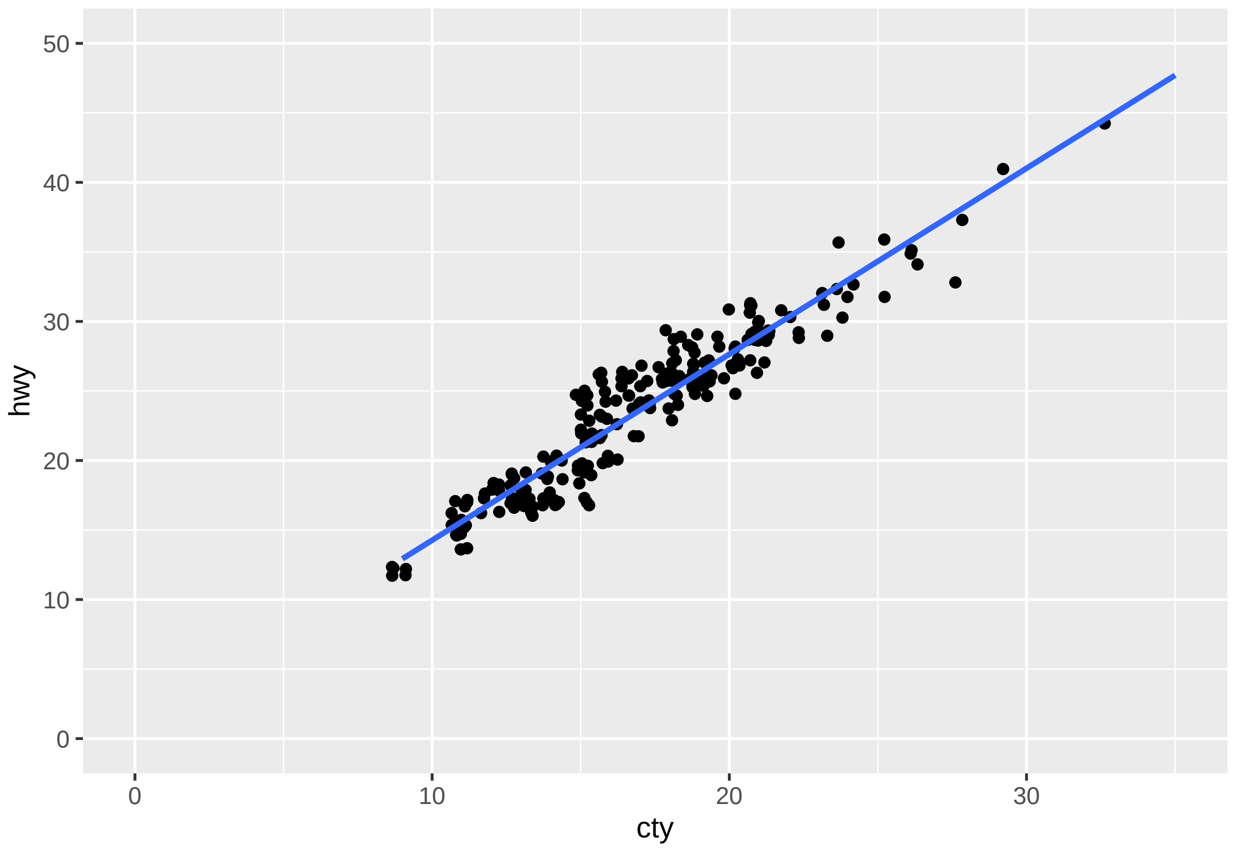
2. How do we get rid of the grey shading from our plot?

Answer: Looking at the help page for `geom_smooth()`, we specify `se=FALSE` inside `geom_smooth()`:

```
mpg %>%
  ggplot(aes(x=cty, y=hwy)) +
  geom_jitter() +
  geom_smooth(method="lm", se=FALSE) +
  xlim(c(0, 35)) +
  ylim(c(0, 50))
```

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

```
## Warning: Removed 1 rows containing missing values (`geom_point()`).
```



Prediction

We can use `predict()` to predict the highway mpg of a car using our regression model.

It is best to pass `predict()` a data frame/tibble (with the same column names as the tibble used in fitting the model). This is true even if you only want a single prediction.

For instance, let's say we want to predict the highway mpg of a car getting 20 mpg in the city. We start with a single 1x1 data frame:

```
new_car <- data.frame(cty=20)
new_car
```

```
##   cty
## 1  20
```

Then we pass this data frame into the `newdata` argument of `predict()` to get our prediction:

```
predict(object=car_model, newdata=new_car)
```

```
##        1
## 27.64115
```

We can verify this prediction manually using the slope and intercept stored in `coefs` above:

```r
coefs[1] + 20*coefs[2]
```

```
## (Intercept)
##    27.64115
```

Of course, we can also get multiple predictions simultaneously:

```r
new_cars <- data.frame(cty=c(20, 15, 32, 19.6))
new_cars
```

```
##    cty
## 1 20.0
## 2 15.0
## 3 32.0
## 4 19.6
```

```r
predict(car_model, newdata=new_cars)
```

```
##        1        2        3        4
## 27.64115 20.95388 43.69062 27.10617
```

If we don't specify the **newdata** argument to predict(), we get all the predictions on the original observations used to fit the model:

```r
mpg %>%
  mutate(predicted_hwy = predict(object=car_model)) %>%
  dplyr::select(cty, hwy, predicted_hwy)
```

```
## # A tibble: 234 x 3
##      cty   hwy predicted_hwy
##    <int> <int>         <dbl>
##  1    18    29          25.0
##  2    21    29          29.0
##  3    20    31          27.6
##  4    21    30          29.0
##  5    16    26          22.3
##  6    18    26          25.0
##  7    18    27          25.0
##  8    18    26          25.0
##  9    16    25          22.3
## 10    20    28          27.6
## # i 224 more rows
```

One thing we might notice is that our regression model will predict a car with 0 cty mpg actually has 0.892 hwy mpg (thanks to the nonzero intercept).

In this setting, we likely want to predict a car getting 0 city mpg to get 0 highway mpg, i.e. we want our regression line to go through (0, 0). It turns out you can force your regression line to go through 0, i.e. force the intercept to be 0. We can do this by adding a `0+` on the right side of the formula:

```r
mpg_fit_no_intercept <- lm(hwy ~ 0+cty, data=mpg)
mpg_fit_no_intercept$coefficients
```

```
##     cty
## 1.38721
```

There's no general statistical guidance for when this should be done. Whether or not to include an intercept in your linear regression model depends on whether it is scientifically correct to predict exactly 0 for an x value of 0.

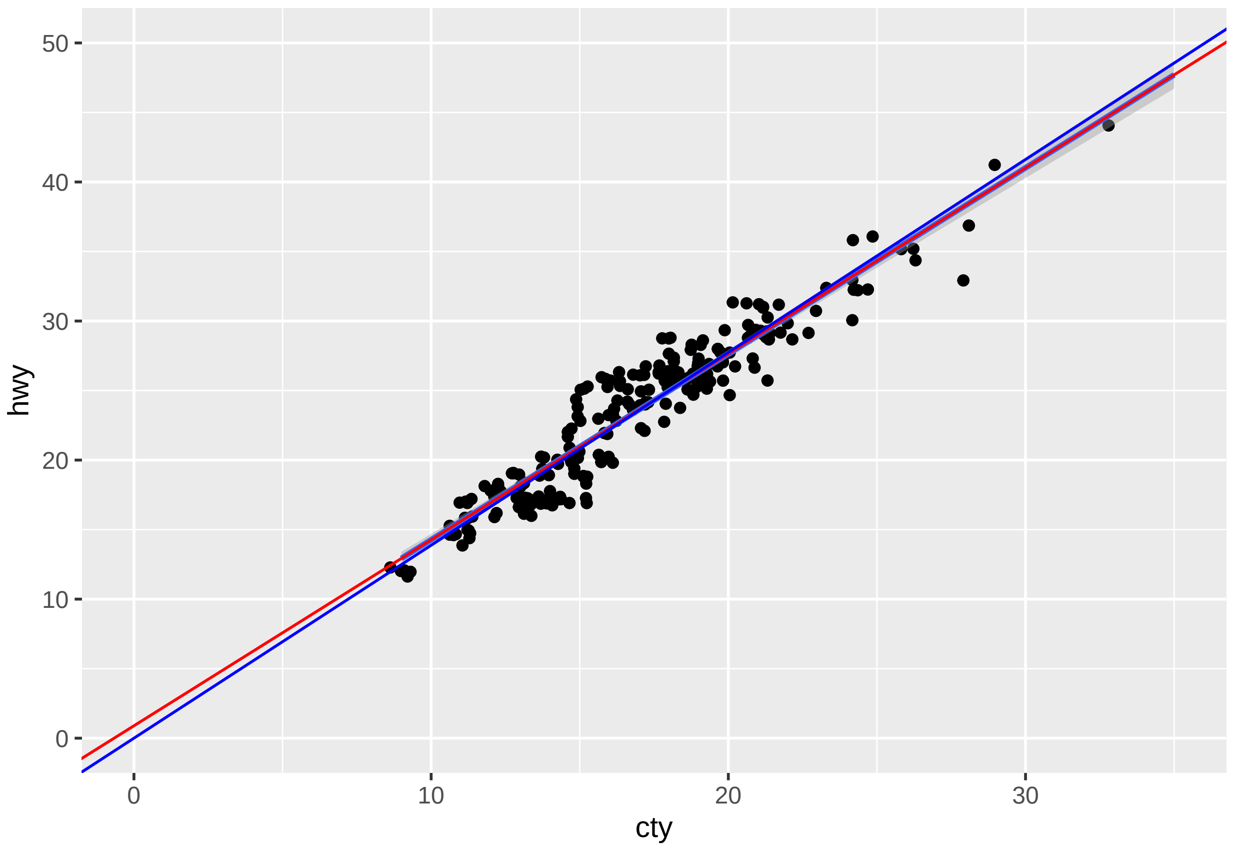
3. Overlay both regression lines — with and without intercept — on the scatterplot above where we demonstrated `geom_smooth()`.

Answer: We use `geom_abline()` to plot the two regression lines. To do so we need the slopes and intercepts of those regression lines, which we extract from the model objects above. Note we only get a slope for the no intercept model, since by definition the intercept there is constrained to 0.

```r
intercept <- car_model$coefficients[1]
slope <- car_model$coefficients[2]
no_intercept_slope <- mpg_fit_no_intercept$coefficients[1]
mpg %>%
  ggplot(aes(x=cty, y=hwy)) +
  geom_jitter() +
  geom_smooth(method="lm") +
  geom_abline(slope=slope, intercept=intercept, color="red") +
  geom_abline(slope=no_intercept_slope, intercept=0, color="blue") +
  xlim(c(0, 35)) +
  ylim(c(0, 50))
```

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

```
## Warning: Removed 1 rows containing missing values (`geom_point()`).
```



We see the fits are quite similar.

4. How much did the prediction for a car getting 20 city mpg change from the with-intercept model to the no-intercept model?

Answer:

```r
prediction <- predict(object=car_model, newdata=tibble(cty=20))
no_intercept_prediction <- predict(object=mpg_fit_no_intercept, newdata=tibble(cty=20))
no_intercept_prediction - prediction
```

```
##         1
## 0.1030535
```

Binary predictors

We now fit a linear regression with a binary predictor. In particular, let's try to predict highway mileage depending on whether the car is 4 wheel drive or 2 wheel drive.

Front-wheel and rear-wheel drive are both forms of 2 wheel drive. Thus, we can create a binary variable `drv_binary` that equals "two" when `drv` is `f` or `r`, and "four" when `drv` is "4" (see the help menu for `mpg` and `ifelse()`):

```r
mpg_2 <- mpg %>%
  mutate(drv_binary=ifelse(drv=="4", "four", "two")) %>%
  select(hwy, drv_binary)
head(mpg_2)
```

```
## # A tibble: 6 x 2
##     hwy drv_binary
##   <int> <chr>
## 1    29 two
## 2    29 two
## 3    31 two
## 4    30 two
## 5    26 two
## 6    26 two
```

Note that I made the values of `drv_binary` "two" and "four" instead of 2 and 4 so that `lm()` recognizes `drv_binary` as categorical, rather than quantitative. You could've also used the numerical versions, and then used `as.character()` or `as.factor()` to convert the type of the column. Alternatively you could've specified values "2" and "4" (as opposed to 2 and 4).

```r
binary_model <- lm(hwy ~ drv_binary, data=mpg_2)
binary_model$coefficients
```

```
##   (Intercept) drv_binarytwo
##     19.174757      7.619136
```

As explained in lecture, by default "four" is the reference level (the one corresponding to "x=0"), since it comes before "two" alphabetically. Hence the prediction for a 4-wheel drive car is 19.175, and 19.175+7.619=26.794 for a 2-wheel drive car.

We could've re-ordered the factor levels as follows:

```r
mpg_2$drv_binary <- fct_relevel(mpg_2$drv_binary, "two", "four")
```

Since we put "two" as the first level in `fct_relevel()`, it will now be the reference level.

```r
binary_model <- lm(hwy ~ drv_binary, data=mpg_2)
binary_model$coef
```

```
##    (Intercept) drv_binaryfour
##      26.793893      -7.619136
```

Note that re-leveling a factor does not change any of the predictions (it would be concerning if it did).

If you didn't know anything about linear regression, a reasonable way to get a hwy mpg prediction based on whether a car is 2 wheel drive or 4 wheel drive would be to simply take the average hwy mpg of all cars in each category:

```
mpg_2 %>%
  group_by(drv_binary) %>%
  summarise(hwy_mean = mean(hwy))
```

```
## # A tibble: 2 x 2
##   drv_binary hwy_mean
##   <fct>         <dbl>
## 1 two            26.8
## 2 four           19.2
```

Whoa! We get the exact same predictions! It turns out that you can show mathematically this is not a coincidence. That is, simple linear regression with a binary predictor yields predictions corresponding to the average dependent variable value for each of the two possible predictor values.

5. Create a binary predictor `trans_type` that is either `auto` or `manual` based on the first few characters in the existing `trans` column. Then fit a simple linear regression to predict `hwy` from `trans_type`. Hint: consider the `substr()` function.

Answer: Recall the `if_else()` construction introduced a few times in the lecture notes.

```
mpg <- mpg %>%
  mutate(trans_type = if_else(substr(mpg$trans, 1, 1) == "a", "auto", "manual"))
summary(lm(hwy ~ trans_type, mpg))
```

```
##
## Call:
## lm(formula = hwy ~ trans_type, data = mpg)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.7792  -5.2930   0.2208   3.7070  18.7070
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       22.2930     0.4578  48.696  < 2e-16 ***
## trans_typemanual   3.4862     0.7981   4.368 1.89e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.736 on 232 degrees of freedom
## Multiple R-squared:  0.076,  Adjusted R-squared:  0.07202
## F-statistic: 19.08 on 1 and 232 DF,  p-value: 1.888e-05
```

Model diagnostics: residual plots

Above, we loosely looked at the scatterplot and "eye-balled" a linear trend to justify the use of a linear regression.

A better, more principled visual diagnostic is to examine a *residual plot.*

The *residuals* from a linear regression fit are, by definition, the actual y values minus the y values predicted by the regression line. For example, our regression model predicts that a car with 20 cty mpg should have

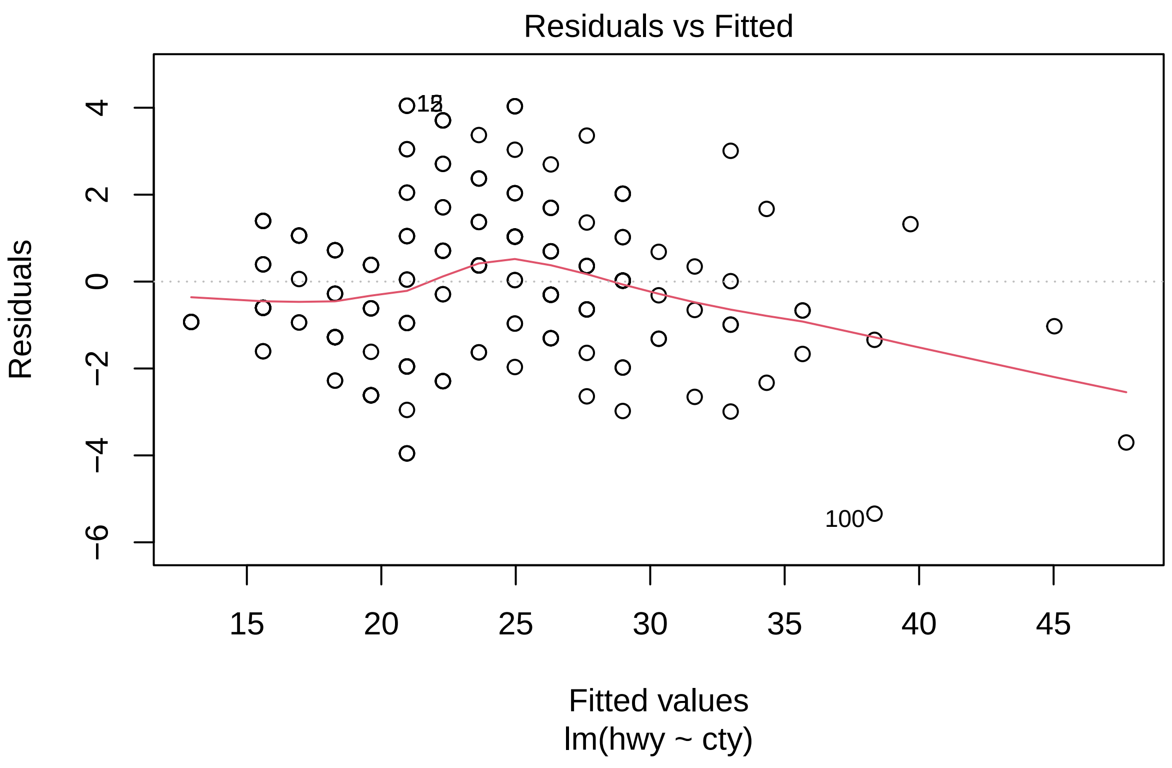
0.892+1.337*20 = 27.632 hwy mpg. If the car actually got 31 hwy mpg, the residual for that car would be 31-27.632 = 3.368.

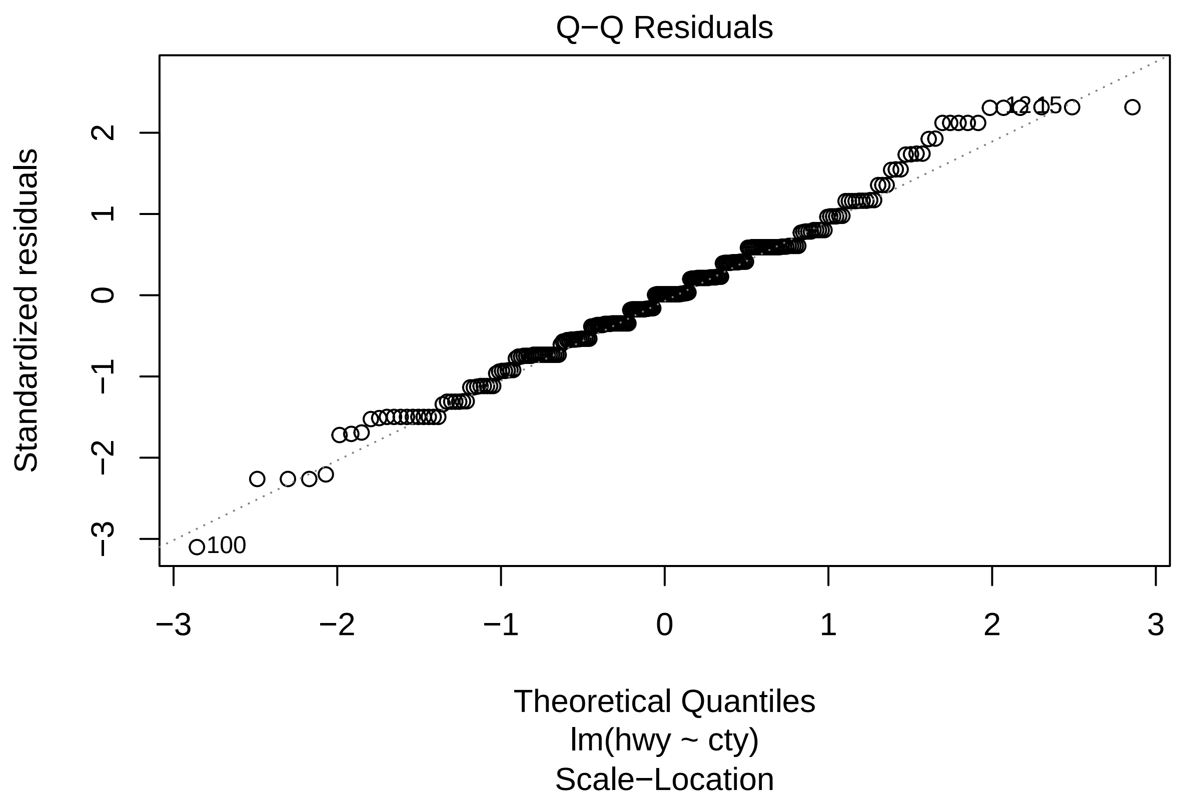
For a linear regression model to be justified, the residuals should show no discernible pattern or trend above or below zero.

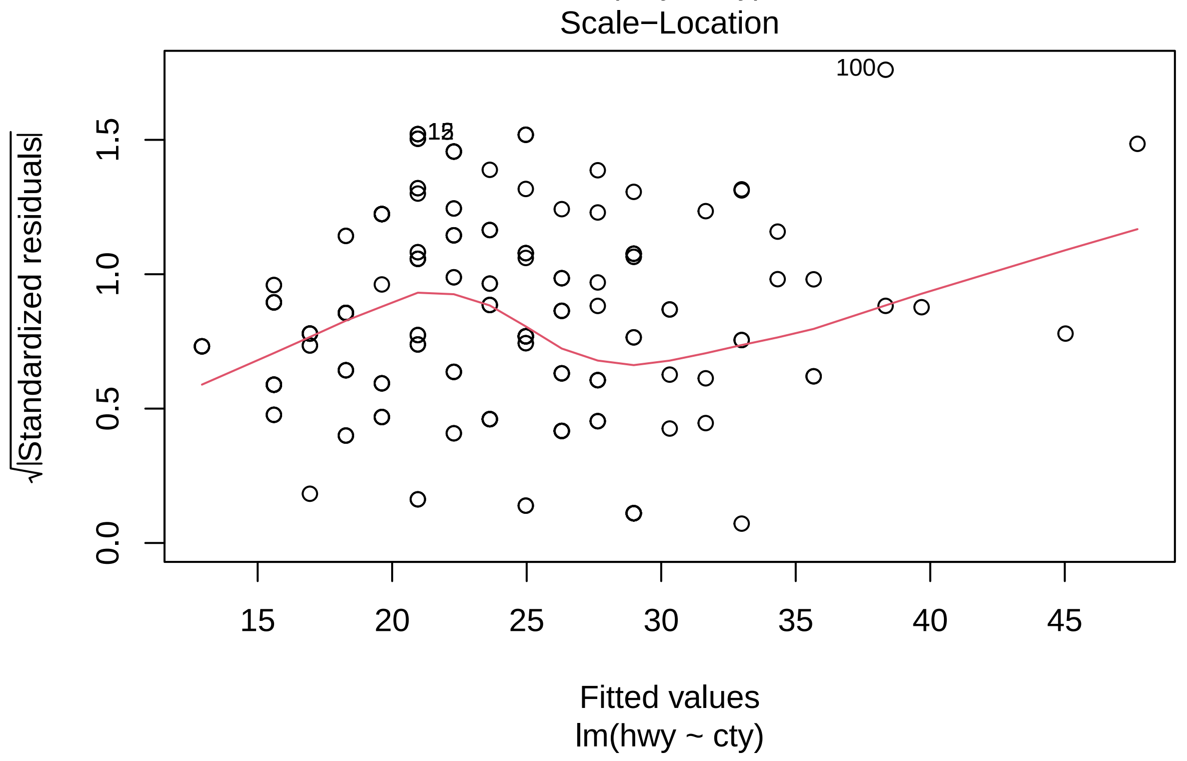
A useful way to check this visually is to call `plot()` on the `lm()` object. You will get a series of 4 different plots. The first one plots “Residuals vs Fitted”, which is a scatterplot of the the residuals on the y-axis versus the regression line predictions on the x axis.

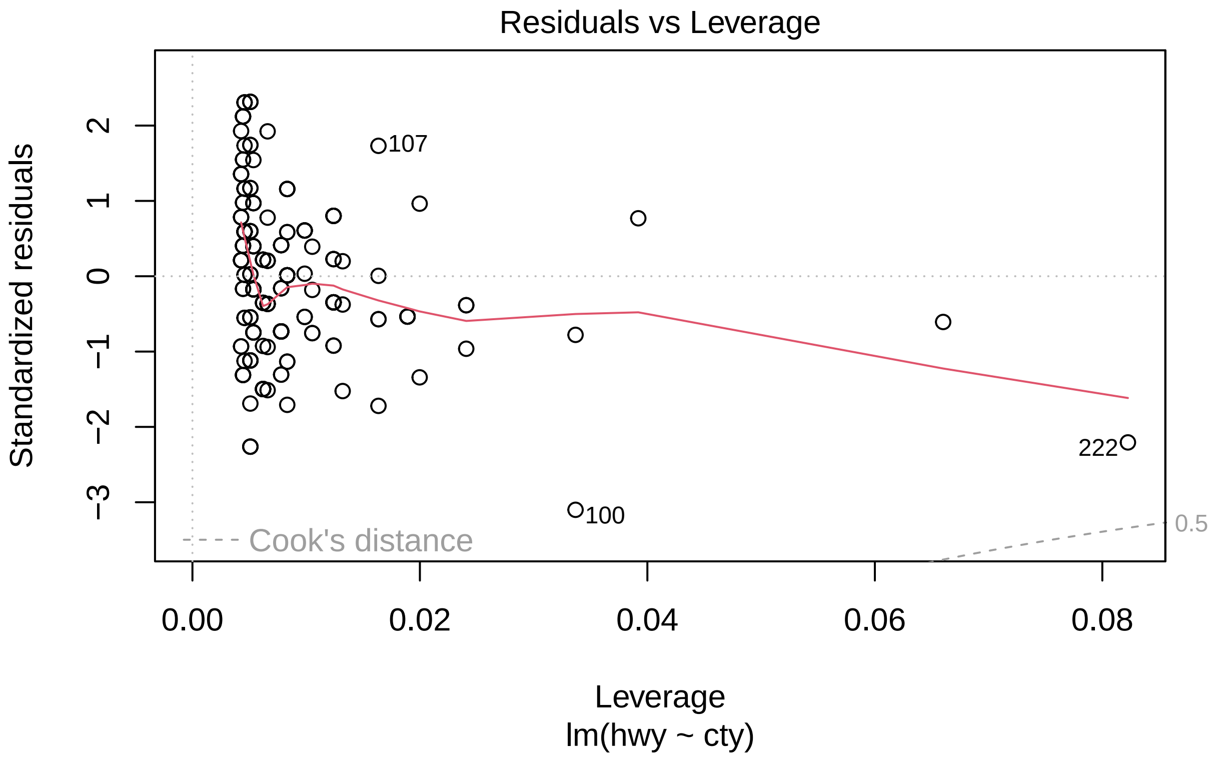
These points should look randomly scattered about 0. For the `car_model` regression above, it seems that there are some patterns in the residuals (e.g. the residuals tend to decrease as the fitted value increases from about 20 to 35), indicating some violation of the assumptions of a linear regression model. We’ll look at more complex models to address this next class.

```r
plot(car_model)
```




Q−Q Residuals
Standardized residuals
Theoretical Quantiles
lm(hwy ~ cty)
100
12
15


Scale−Location
√|Standardized residuals|
Fitted values
lm(hwy ~ cty)
100
12
15


Residuals vs Leverage
Standardized residuals
2
1
0
−1
−2
−3
107
222
100
Cook's distance
0.5
0.00
0.02
0.04
0.06
0.08
Leverage
lm(hwy ~ cty)