Lecture 10

Peter Shaffery

2/16/2021

More Tidyverse: `reshape2`

We've already learned about concept of "tidy" data, recall this example from Week 2:

Untidy Data

year_2010	year_2011	year_2012
10	14	2
13	9	5
8	12	4

Tidy Data

year	counts
2010	10
2010	13
2010	8
2011	14
2011	9
2011	12
2012	2
2012	5
2012	4

Tidy data is really important for working within the tidyverse effectively.

What do you do when you get an untidy dataset? One tool is a package called `reshape2`.

```
library(tidyverse)
library(magrittr)
library(broom)

untidy.df = data.frame(
  'year_2010'=c(10,13,8),
  'year_2011'=c(14,9,12),
```

```
  'year_2012'=c(2,5,4)
  )
untidy.df
```

```
##   year_2010 year_2011 year_2012
## 1        10        14         2
## 2        13         9         5
## 3         8        12         4
```

`reshape2` provides a number of useful tools for changing the *structure* of a dataset (eg. going from untidy to tidy data, or back again).

One of the core tools it provides for this is a function called `melt`:

```
tidy.df = reshape2::melt(untidy.df)
```

```
## No id variables; using all as measure variables
```

```
tidy.df
```

```
##    variable value
## 1 year_2010    10
## 2 year_2010    13
## 3 year_2010     8
## 4 year_2011    14
## 5 year_2011     9
## 6 year_2011    12
## 7 year_2012     2
## 8 year_2012     5
## 9 year_2012     4
```

The opposite of `melt` is `cast` (either `dcast` for a dataframe or `acast` for an array):

```
tidy.df$id = 1:3
tidy.df %>% reshape2::dcast(formula = id~variable) # formula = row~column
```

```
##   id year_2010 year_2011 year_2012
## 1  1        10        14         2
## 2  2        13         9         5
## 3  3         8        12         4
```

Let's see a example with real data:

```
library(reshape2)
names(airquality) %<>% tolower
airquality %<>% drop_na

airquality %>% head
```

```
##   ozone solar.r wind temp month day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    23     299  8.6   65     5   7
## 6    19      99 13.8   59     5   8
```

```
airquality %>% melt %>% head
```

```
##   variable value
## 1    ozone    41
## 2    ozone    36
## 3    ozone    12
## 4    ozone    18
## 5    ozone    23
## 6    ozone    19
```

What if we didn't want to completely "tall-ify" the data?

```
airquality %>% melt(id.vars=c('month','day')) %>% head
```

```
##   month day variable value
## 1     5   1    ozone    41
## 2     5   2    ozone    36
## 3     5   3    ozone    12
## 4     5   4    ozone    18
## 5     5   7    ozone    23
## 6     5   8    ozone    19
```

Let's see how to use `dcast` here:

```
aq.tall = airquality %>% melt(id.vars=c('month','day'))

aq.tall %>% dcast(month+day ~ variable) %>% head
```

```
##   month day ozone solar.r wind temp
## 1     5   1    41     190  7.4   67
## 2     5   2    36     118  8.0   72
## 3     5   3    12     149 12.6   74
## 4     5   4    18     313 11.5   62
## 5     5   7    23     299  8.6   65
## 6     5   8    19      99 13.8   59
```

For more detailed walk-through of what's going on here, check out: https://seananderson.ca/2013/10/19/reshape/

Transformations

Recall that linear regression relies on a few assumptions:

1. Linearity function
2. Normality of error
3. Homoskedasticity

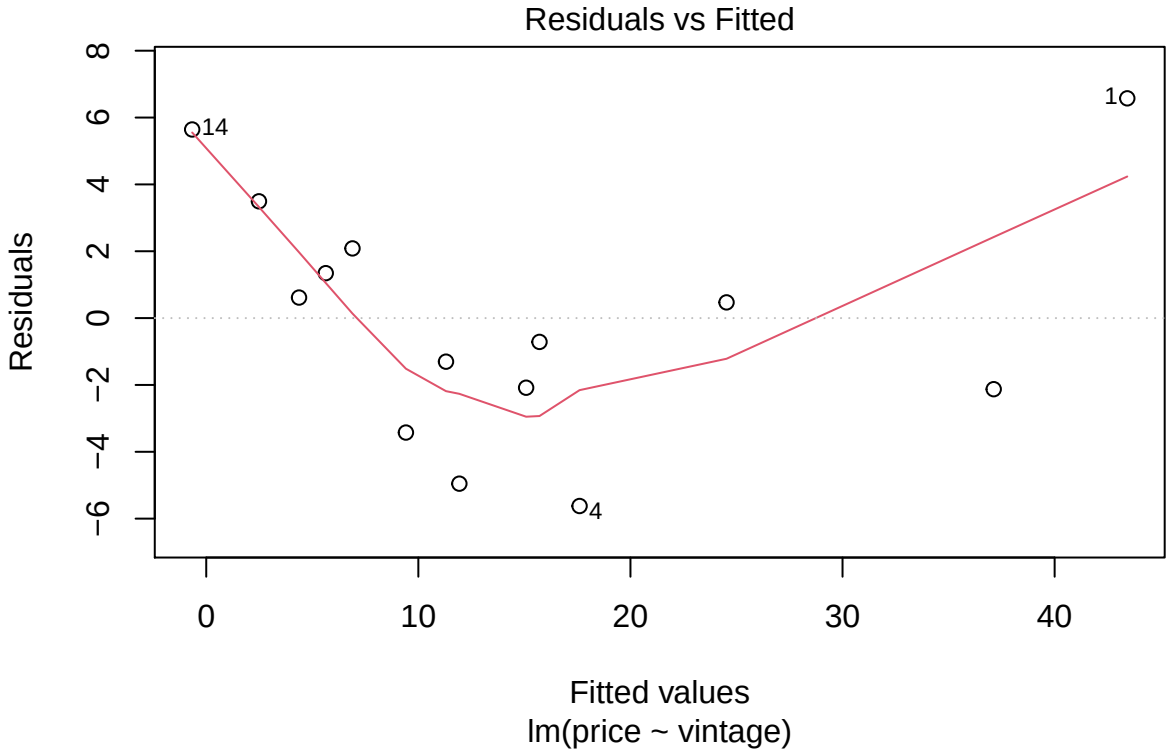
(Do you recall the priority of these assumptions?)

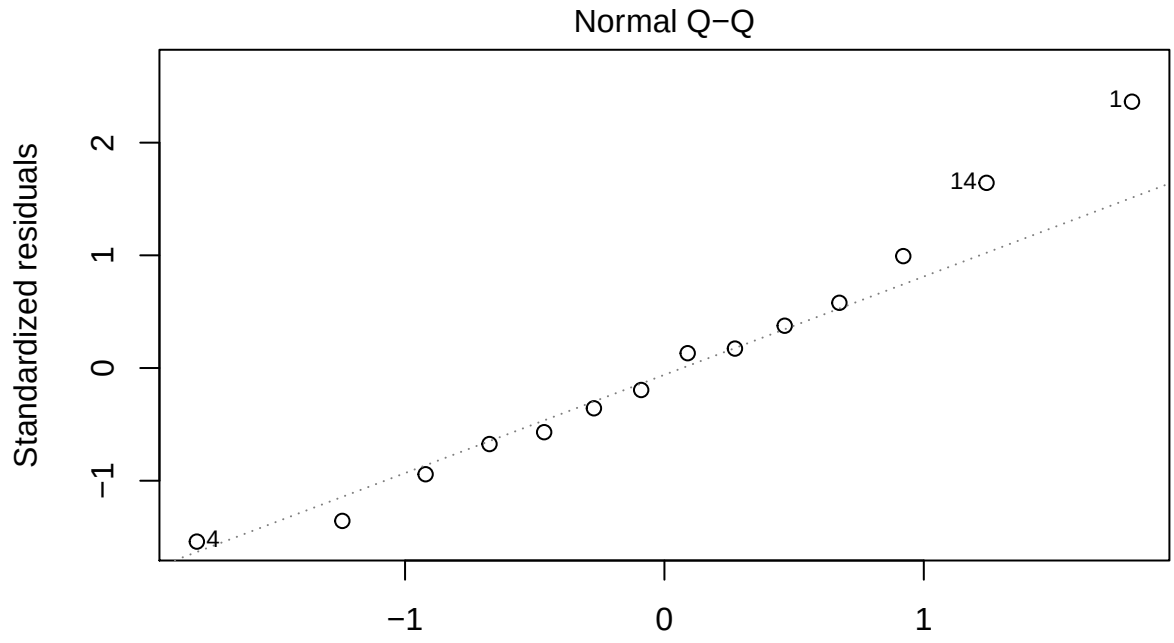
We have previously seen some ways that we can diagnose if these assumptions are met:

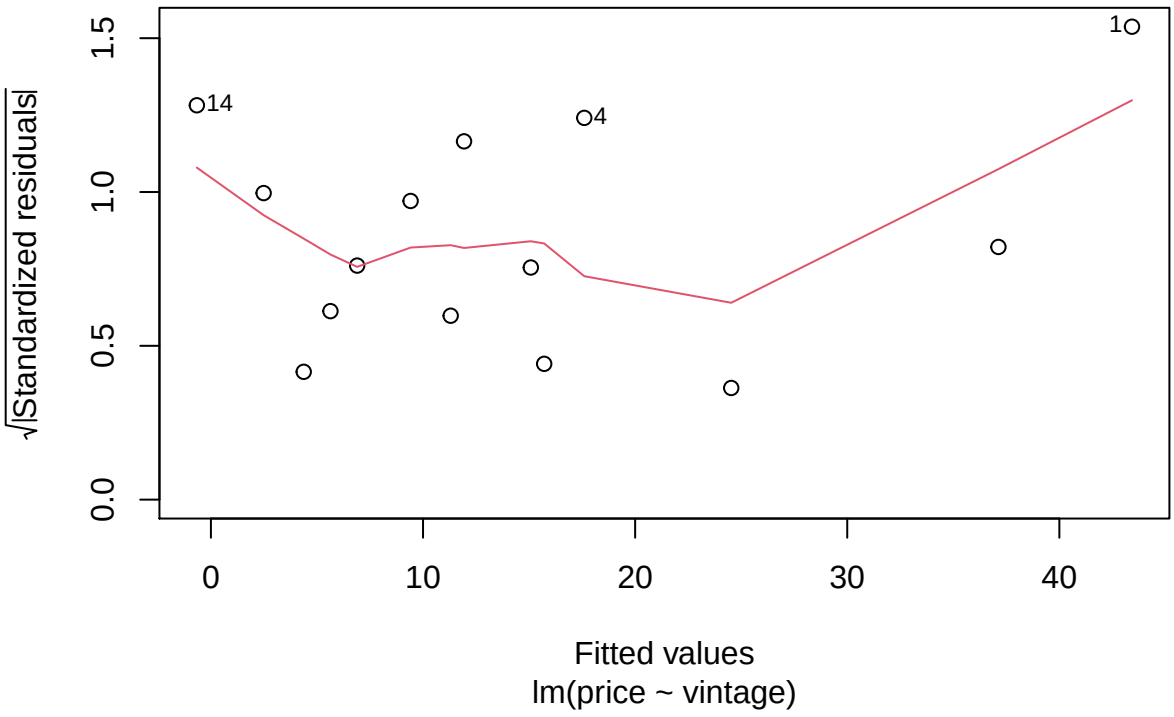
Example: Wine

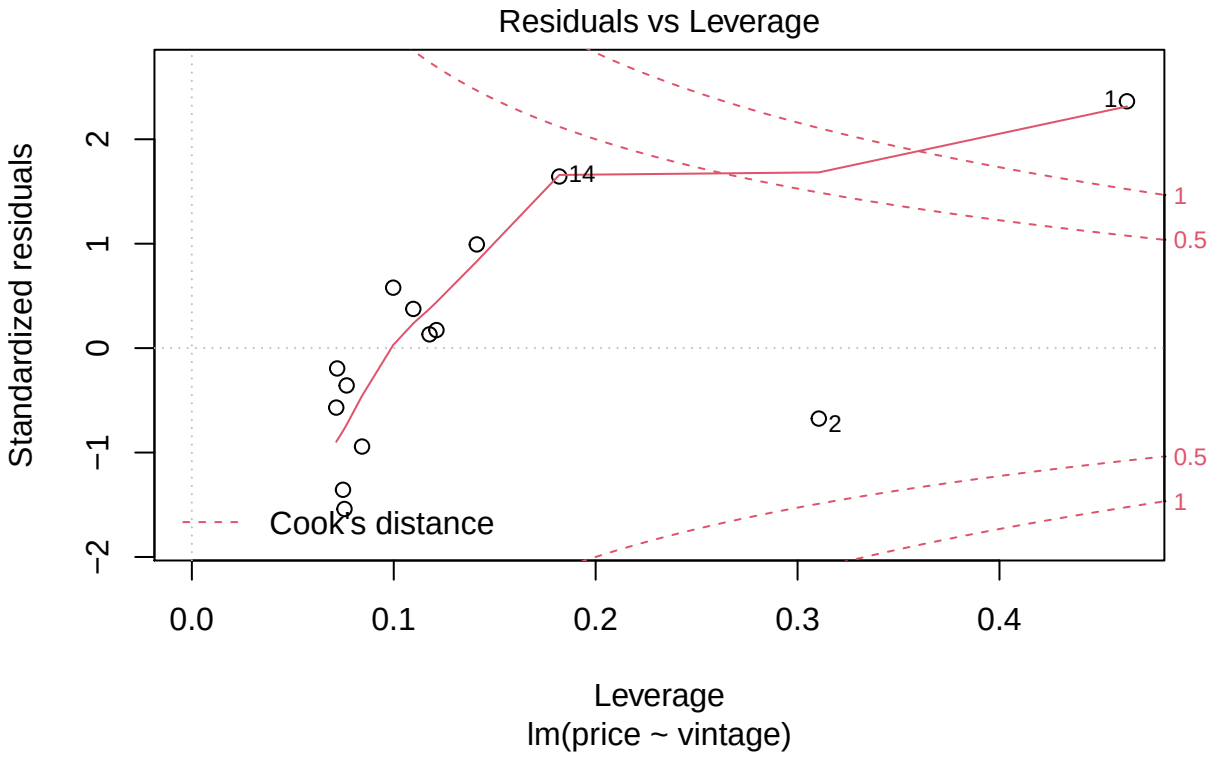
```
wine = read.csv('../../data/wine.csv')

plot(wine$vintage, wine$price,xlab='VINTAGE',ylab='PRICE')
```


```
mod = lm(price~vintage, data=wine)
plot(mod)
```




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(price ~ vintage)
1
14
4
−1
0
1
2


Scale−Location
√|Standardized residuals|
Fitted values
lm(price ~ vintage)
1
14
4
0.0
0.5
1.0
1.5
0
10
20
30
40



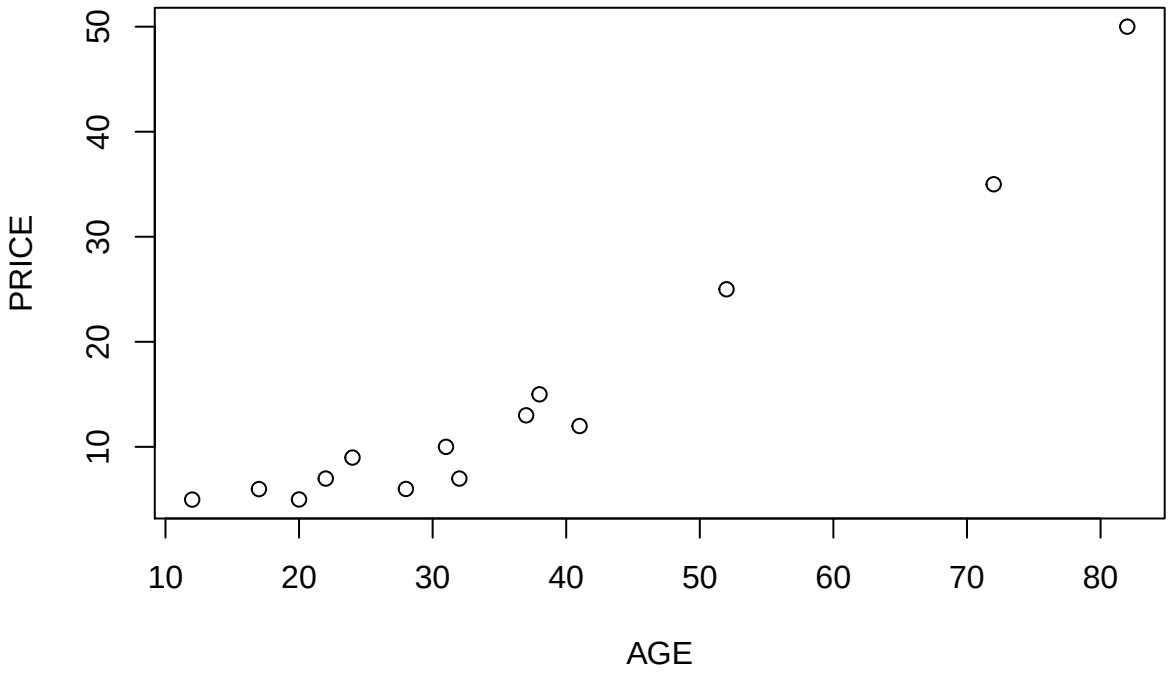
But what can we actually do about it when they are not met? For some problems we can *transform* the data, for example work with $\ln(y)$ instead of y.

Linearizing Transforms

Function	Transform	Linear Form
$y = ax^\beta$	$y' = \log(y)$, $x' = \log(x)$	$y' = \log a + \beta x'$
$y = ae^{\beta x}$	$y' = \ln(y)$	$y' = \ln a + \beta x$
$y = \alpha + \beta \ln(x)$	$x' = \ln(x)$	$y = \alpha + \beta x'$
$y = \frac{x}{\alpha x - \beta}$	$y' = \frac{1}{y}$, $x' = \frac{1}{x}$	$y' = \alpha - \beta x'$

Let's try applying this to the wine example. Before we do anything, let's reparameterize our x-axis in terms of AGE rather than VINTAGE:

```
wine$age = 1972 - wine$vintage

plot(wine$age,wine$price,xlab='AGE',ylab='PRICE')
```



Since the graph appears to have a little curve upwards, let's say:

$$\text{PRICE} = \beta_0 \exp\left[\beta_1 \text{AGE}\right]$$

From the table, this suggests the transform:

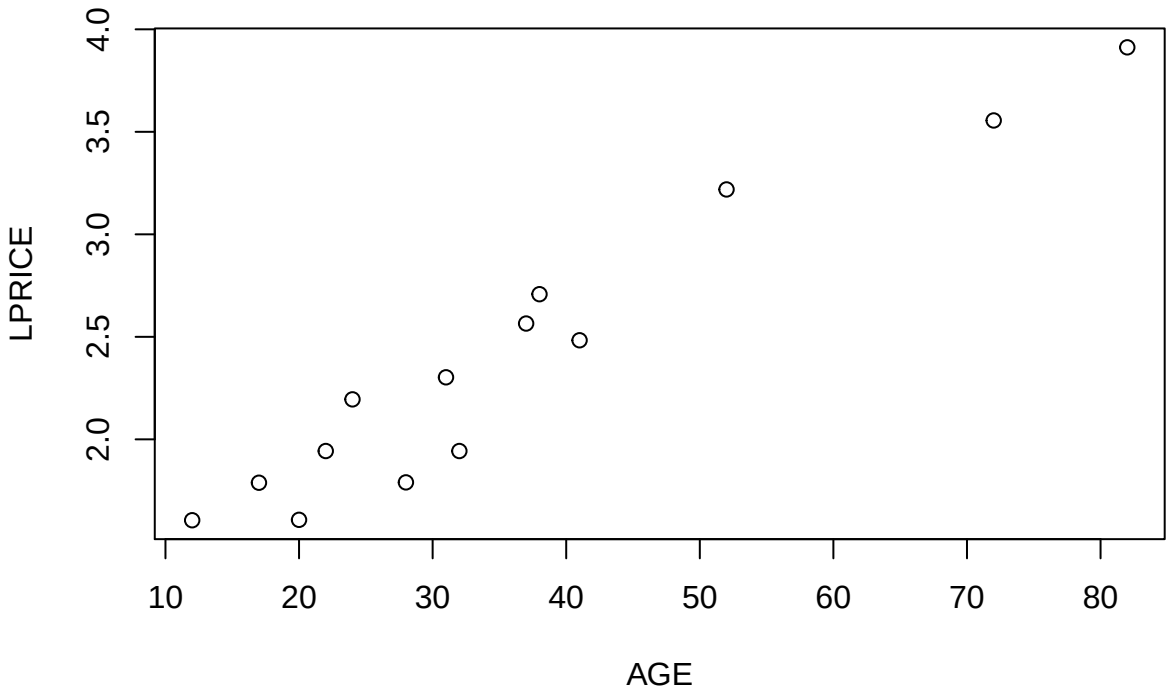
$$\text{LPRICE} = \ln\left[\text{PRICE}\right]$$

And so our model would be:

$$LPRICE = \ln \beta_0 + \beta_1 \text{AGE}$$

Let's just quickly visualize the effect of this transform:

```
plot(wine$age,log(wine$price),xlab='AGE',ylab='LPRICE')
```



Way better!

To clarify, our regression model is now:

$$E[\log(PRICE)] = \beta_0 + \beta_1 \text{AGE}$$

Thus every unit increase of AGE (for every year the wine gets older) increases $\log(\text{PRICE})$ by β_1 log-dollars.

What does this mean on the scale of dollars?

We might try:

$$\exp(E[\log(\text{PRICE})]) = \exp(\beta_0)\exp(\beta_0 \text{AGE})$$

And this isn't *wrong*, but it's also true that:

$$\exp(E[\log(\text{PRICE})]) \neq E[\text{PRICE}]$$

Indeed it's actually the case that:

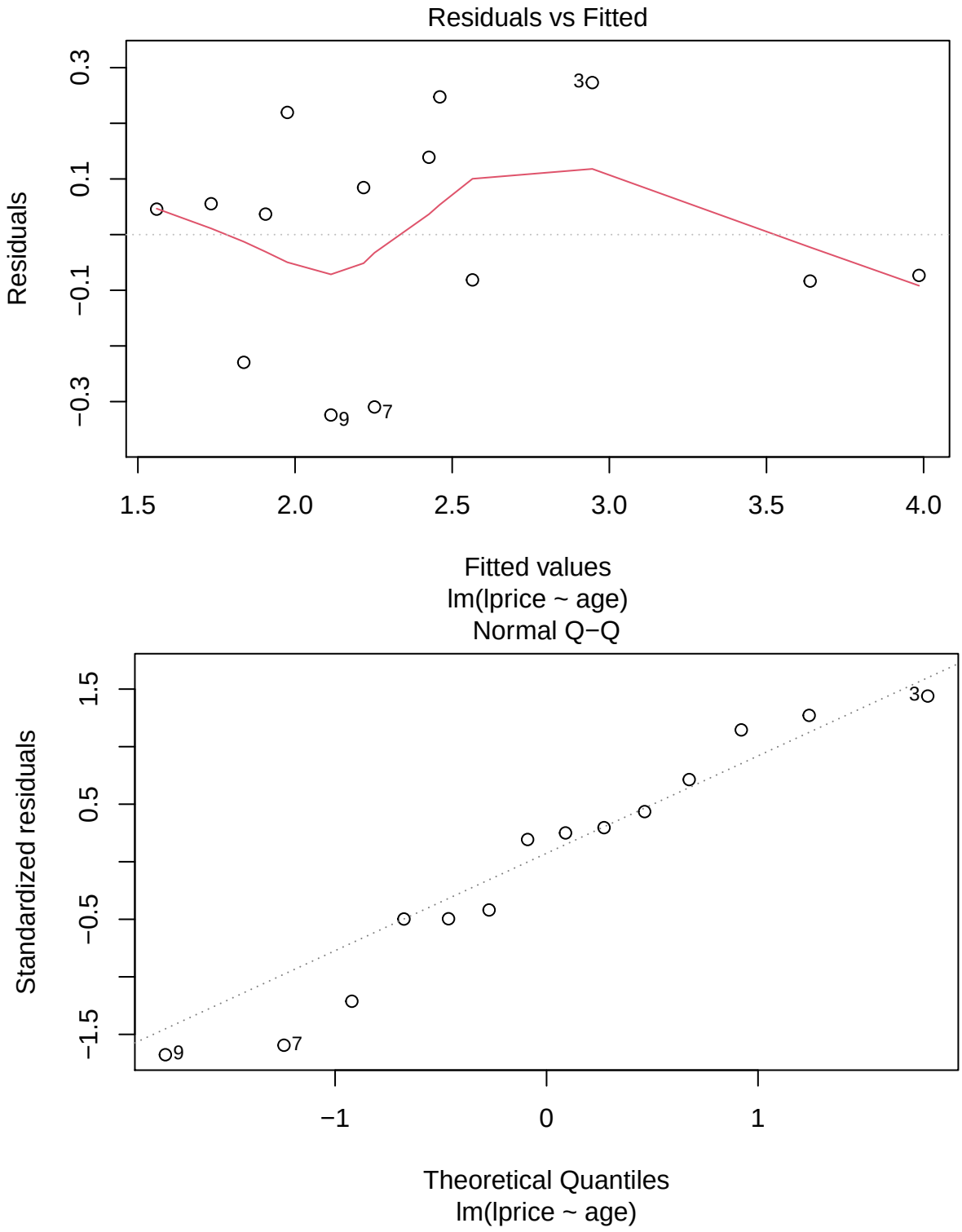
$$\exp(E[\log(\text{PRICE})]) \leq E[\text{PRICE}]$$

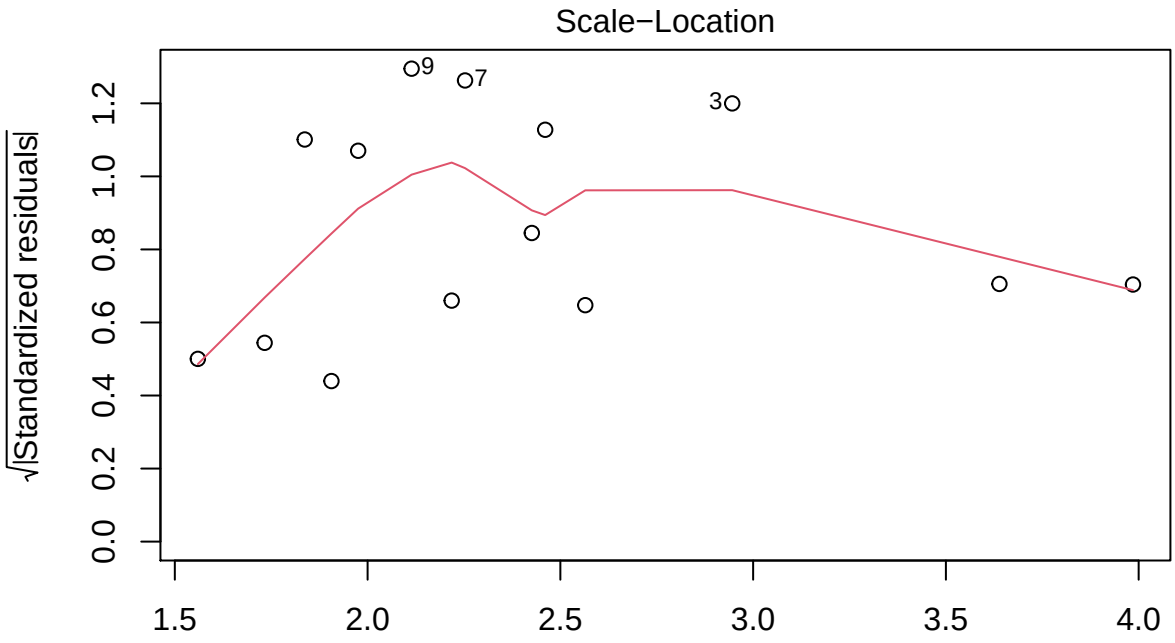
Which is a result of Jensen's Inequality.

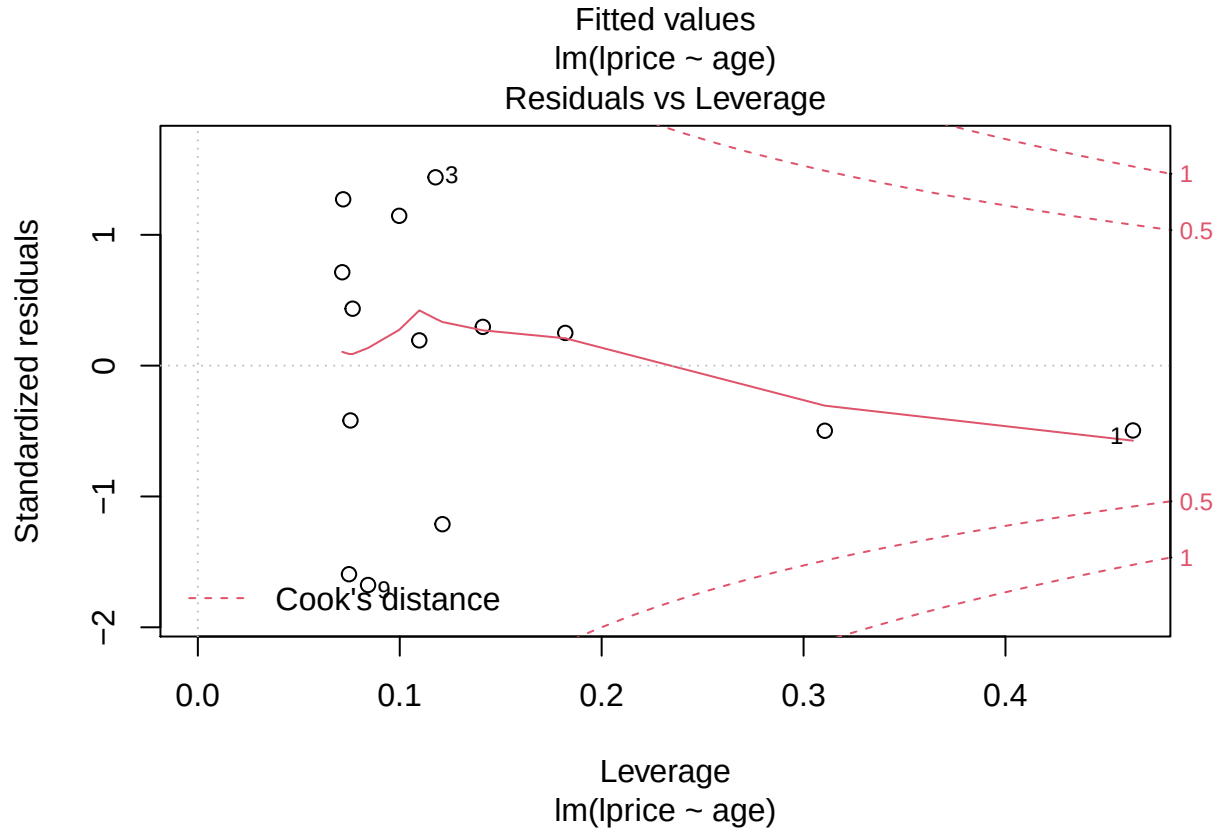
So exponentiating both sides doesn't *technically* allow us to talk about the average effect of AGE on PRICE (like we're used to in SLR and MLR). We'll come back to this in just a second.

Despite any weirdness with interpretation, the back-transformed model turns out pretty good predictions:

```
wine$lprice = log(wine$price)
mod.linear = lm(price~age, data=wine)
mod.log = lm(lprice~age, data=wine)

plot(mod.log)
```

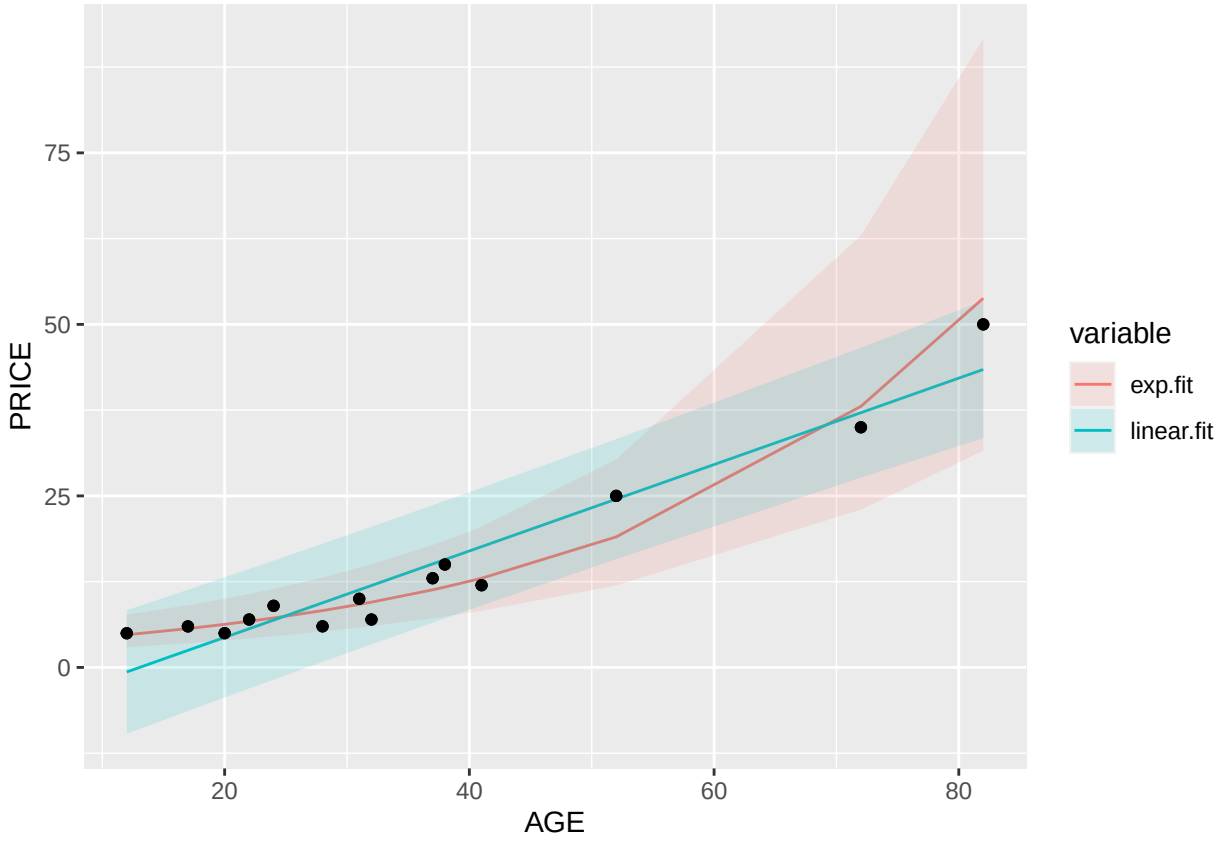

Residuals vs Fitted
Residuals
0.3
0.1
−0.1
−0.3
3
9
7
1.5
2.0
2.5
3.0
3.5
4.0
Fitted values
lm(lprice ~ age)
Normal Q−Q
Standardized residuals
1.5
0.5
−0.5
−1.5
3
9
7
−1
0
1
Theoretical Quantiles
lm(lprice ~ age)






```
plot.df = mod.linear %>%
  augment(interval='prediction') %>%
  select(age,.fitted,.lower,.upper) %>%
  rename('linear.fit'=.fitted,'upper'=.upper,'lower'=.lower) %>%
  reshape2::melt(id.vars=c('age','upper','lower'))
```

```
plot.df = mod.log %>%
  augment(interval='prediction') %>%
  mutate('.fitted'=exp(.fitted),'lower'=exp(.lower),'upper'=exp(.upper)) %>%
  select(age,.fitted,lower,upper) %>%
  rename('exp.fit'=.fitted) %>%
  reshape2::melt(id.vars=c('age','upper','lower')) %>%
  rbind(plot.df)

plot.df['price'] = wine$price

ggplot(plot.df, aes(x=age,)) +
  geom_line(aes(y=value,color=variable)) +
  geom_ribbon(aes(ymin=lower,ymax=upper,fill=variable), alpha=.15) +
  geom_point(aes(y=price)) +
  labs(x='AGE',y='PRICE')
```



Notice that the prediction interval for the exponential model is assymetric. Hmm, I wonder why...

Here's Why

One thing we've glossed over is how the error term plays into all this. Recall that we fit the regression:

$$\text{LPRICE}_i = \beta_0 + \beta_1 \text{AGE}_i + \epsilon_i$$

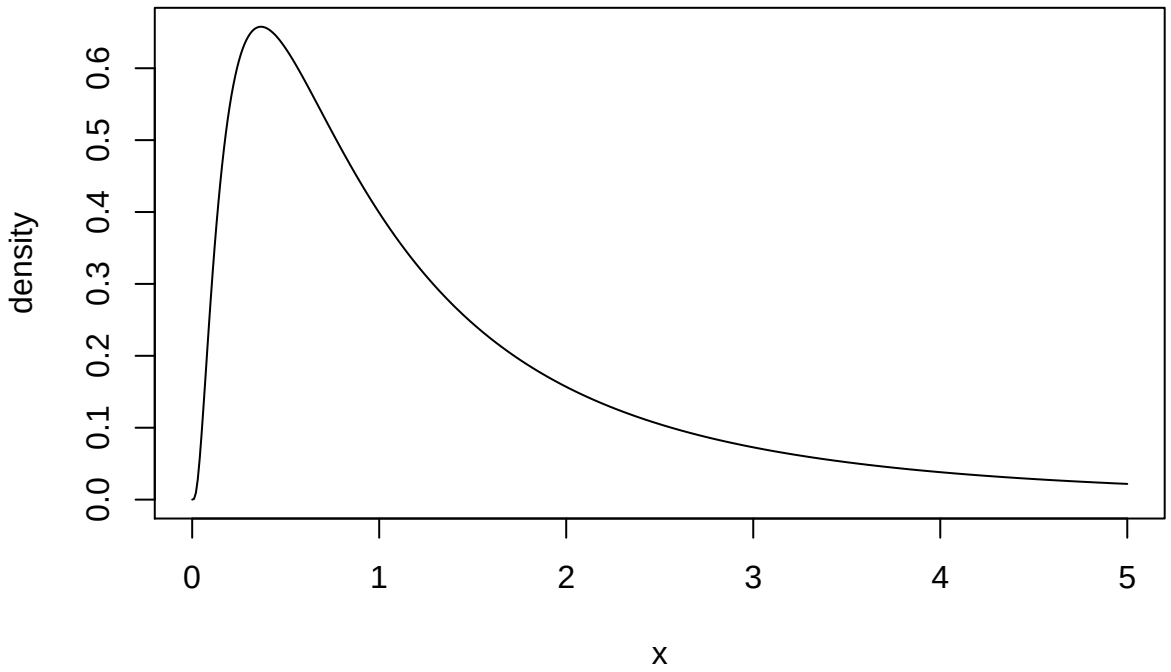
So when we convert *back* to the original scale:

$$\exp\left(\text{LPRICE}_i\right) = \text{PRICE}_i = \exp\left(\beta_0 + \beta_1 \text{AGE}_i\right) \exp(\epsilon_i)$$

So the errors are multiplicative! This explains two things.

First, it (kind of) explains the skew in our prediction intervals. The exponential of a normal random variable (ϵ_i) is called a *log-normal* and its density looks like this:

```
x = seq(0,5,.01)
density = dlnorm(x)
plot(x,density,type='l')
```



So the asymmetry in our prediction appears to reflect this new type of error distribution.

Second, it allows us to interpret β_1. Recall that we were unsure whether we could talk about the *average* effect of AGE on PRICE. It turns out that instead we are dealing with the **median** of PRICE. This comes from two observations:

1. If X is a random variable, and $f(X)$ is a *monotonic* function of X (that is, f only increases or decreases in X), then the median of $f(X)$ is just $f(\tilde{X})$ where $\tilde{X}$ is the median of X.
2. Since exp is monotonic, the median of $\exp(\epsilon_i) = \exp(\tilde{\epsilon_i})$ where $\tilde{\epsilon_i}$ is the median of the error distribution. Since the error distribution is normal, with mean zero, then $\tilde{\epsilon_i} = 0$, hence the median of $\exp(\epsilon_i) = 1$.

Rolling this all up we get:

$$\begin{aligned}\text{Median}\left[\text{PRICE}_i\right] &= \text{Median}\left[\exp\left(\beta_0 + \beta_1 \text{AGE}_i\right)\exp(\epsilon_i)\right] \\ &= \exp\left(\beta_0 + \beta_1 \text{AGE}_i\right)\text{Median}\left[\exp(\epsilon_i)\right] \\ &= \exp\left(\beta_0 + \beta_1 \text{AGE}_i\right)1\end{aligned}$$

So our interpretation here is that for a unit increase in a wine's AGE, the median PRICE increases by β_1.

I also want to draw your attention here to the fact that this transformation **assumes the errors are multiplicative**. While this appears to work out okay in this example, it won't always.

Say for example that we instead believed that the errors should *not* be multiplicative, such as in the model:

$$\text{PRICE}_i = \exp\left(\beta_0 + \beta_1 \text{AGE}_i\right) + \epsilon_i$$

Here a clever transformation will not save us. We are instead dealing with something called a **generalized linear model**, and we will talk about this very soon...

Box-Cox

For the wine example, we kind of picked the log-transformation out of nowhere. It happened to fit okay, but there wasn't a ton of careful thought put into the choice.

A standard (and effective) choice for picking a transformation is to look at a handful of transformations that seem reasonable, and keep the one which produces the nicest residuals.

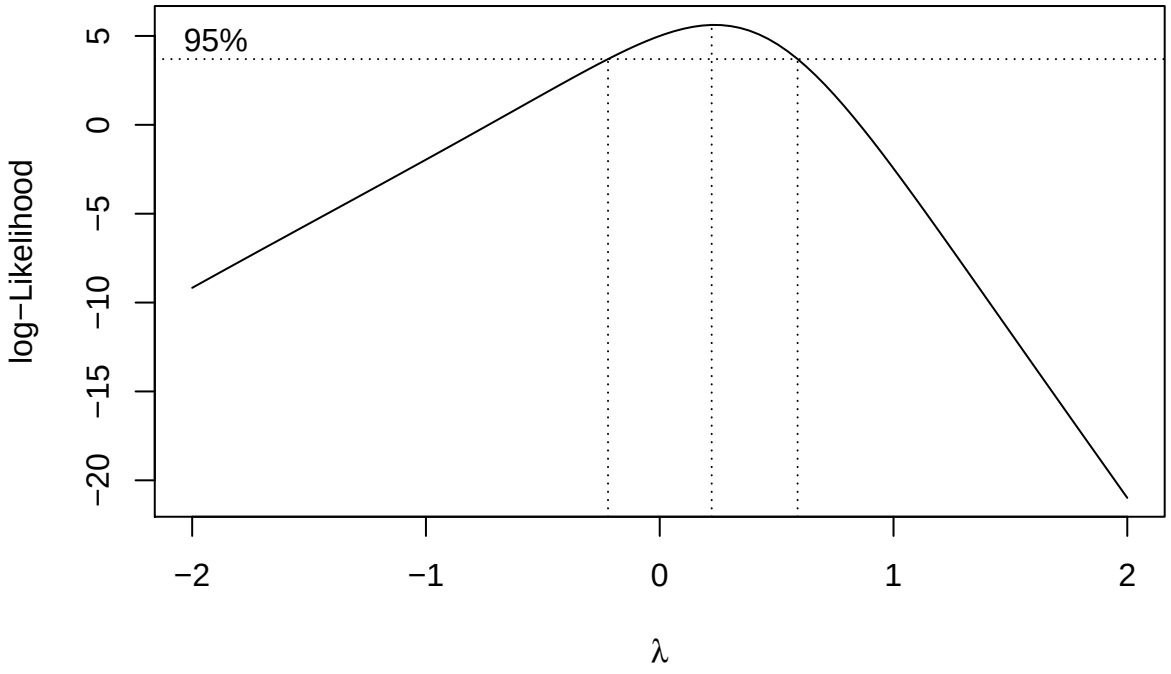
A slightly fancier method to get the **best** transformation is the Box-Cox transform, the idea is that we introduce a new parameter λ, which controls how we transform the dependent variable:

$$t_\lambda(y) = \begin{cases} \frac{y^\lambda - 1}{\lambda}, \lambda \neq 0 \\ \log y, \lambda = 0 \end{cases}$$

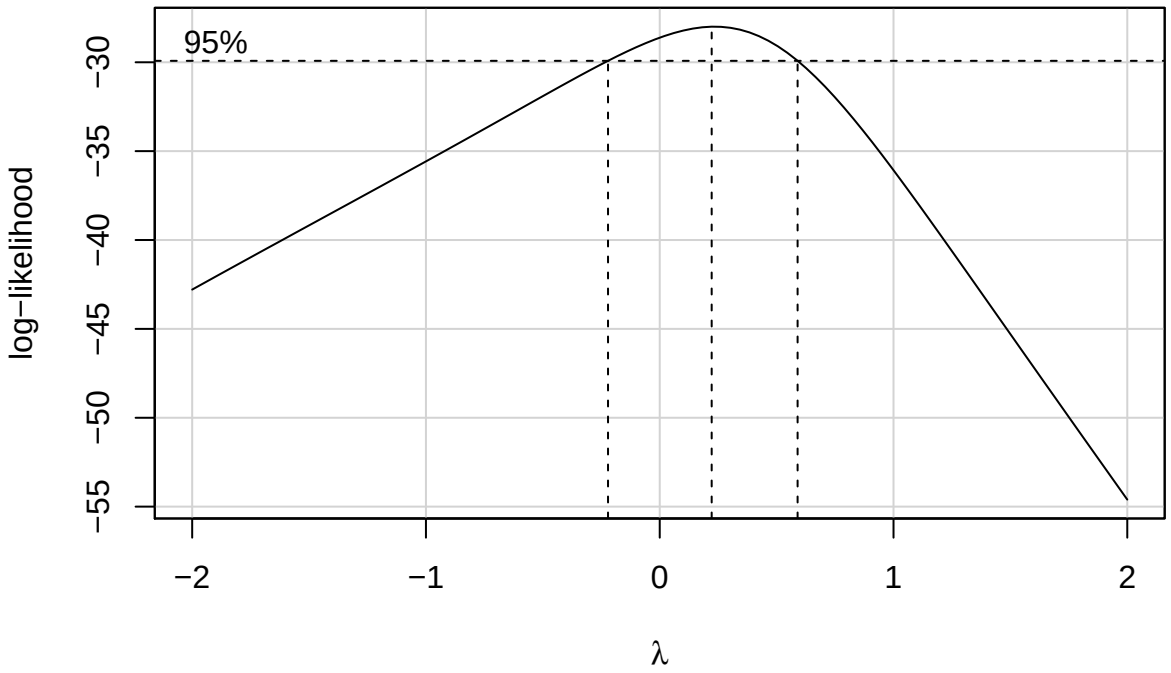
This allows us to compare different possible transformations just by varying λ, rather than having to cook up a bunch of candidate transformations.

Furthermore, just like with β_0 and β_1,we can choose the *best* λ using *maximum likelihood*. We'll skip over the theory of this maximization, as in practice you will basically always just chuck it into a function provided by one of many packages:

```
bc = MASS::boxcox(mod.linear)
```




```
car::boxCox(mod.linear)
```




```
lambda = bc$x[which.max(bc$y)]
lambda
```

```
## [1] 0.2222222
```

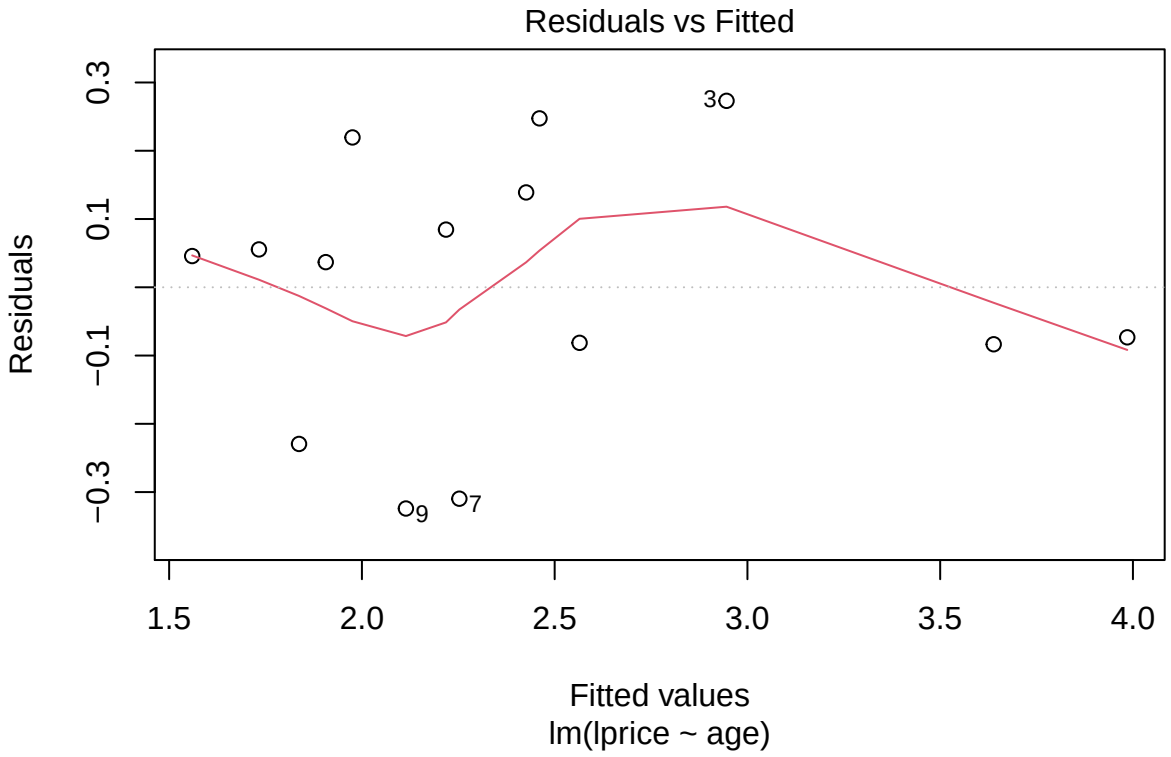
```
# Box-Cox the PRICE variable
bc.func = function(l,x){((x^l) -1 )/l}
wine$price.bc = bc.func(lambda, wine$price)

mod.bc = lm(price.bc~age,data=wine)

mod.log %>% summary
```

```
##
## Call:
## lm(formula = lprice ~ age, data = wine)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.32405 -0.08294  0.04127  0.12533  0.27310
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.143891   0.113931   10.04 3.43e-07 ***
## age         0.034652   0.002765   12.53 2.98e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.202 on 12 degrees of freedom
## Multiple R-squared:  0.929,  Adjusted R-squared:  0.9231
## F-statistic:   157 on 1 and 12 DF,  p-value: 2.978e-08
```

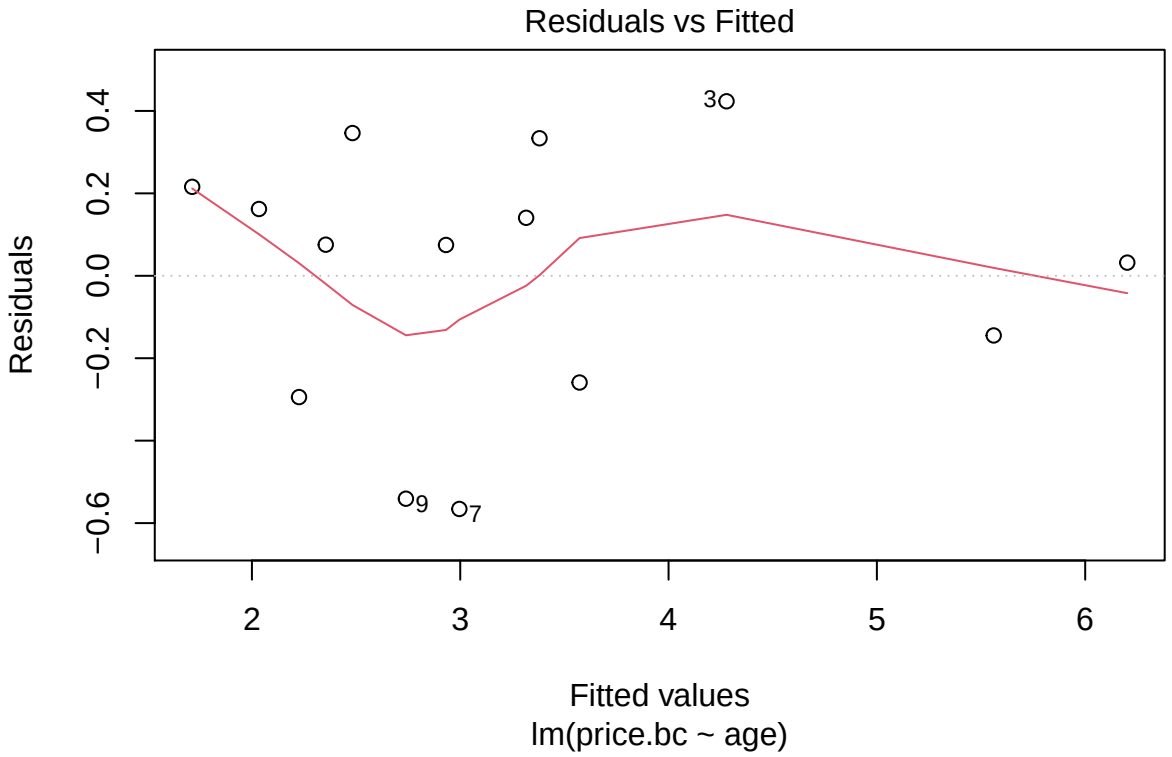
```
mod.bc %>% summary
```

```
##
## Call:
## lm(formula = price.bc ~ age, data = wine)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.56572 -0.23039  0.07517  0.20242  0.42340
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.943769   0.185866   5.078 0.000272 ***
## age         0.064125   0.004511  14.215 7.17e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3295 on 12 degrees of freedom
## Multiple R-squared:  0.9439, Adjusted R-squared:  0.9393
## F-statistic: 202.1 on 1 and 12 DF,  p-value: 7.173e-09
```

```
plot(mod.log, which=1)
```

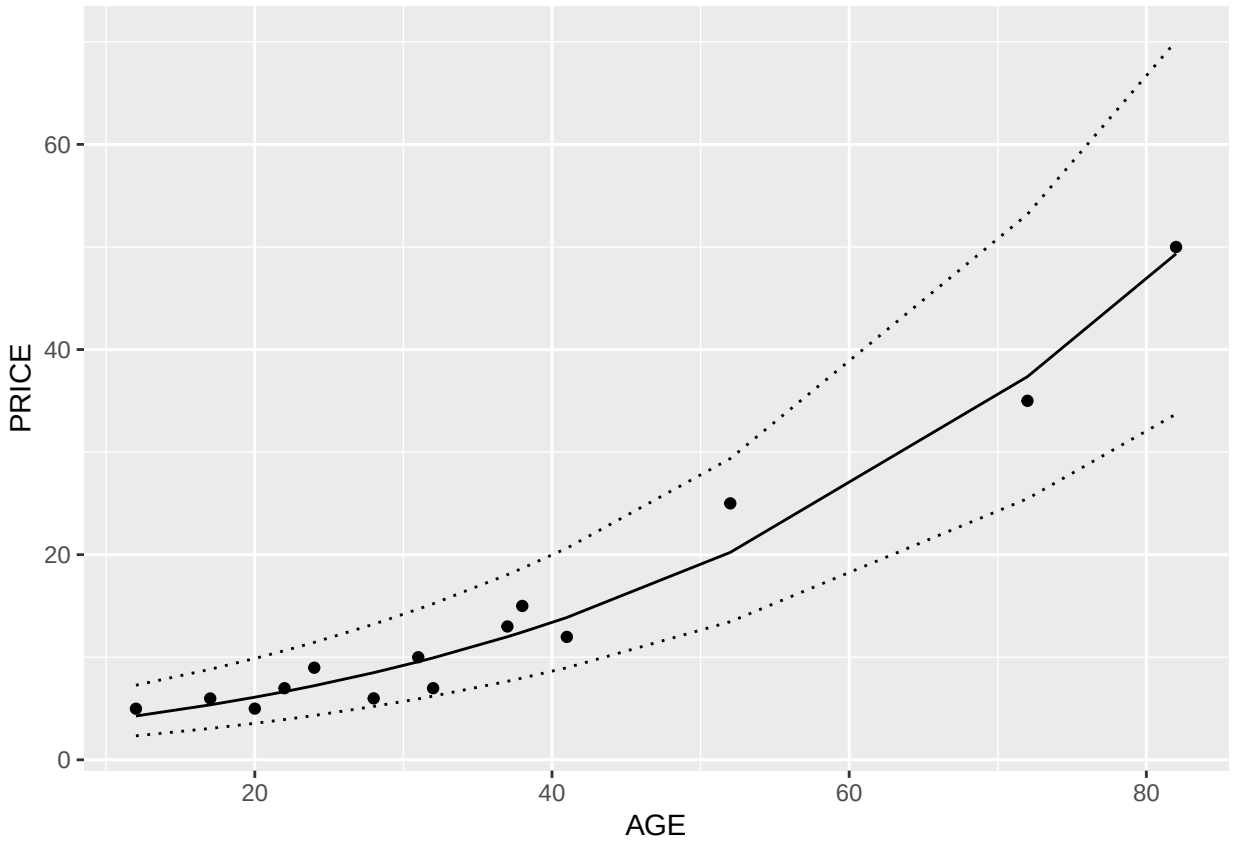



```
plot(mod.bc, which=1)
```



Prediction is similar to the log-transform case, although there's a little bit of algebra needed to invert t_λ

```
inv.bc.func = function(x){ (x*lambda+1)^(1/lambda) }

plot.df = mod.bc %>%
  augment(interval='prediction') %>%
  select(c(.fitted,.lower,.upper)) %>%
  mutate_all(inv.bc.func)

plot.df %<>% cbind(wine[,c('age','price')])

ggplot(plot.df,aes(x=age)) +
  geom_point(aes(y=price)) +
  geom_line(aes(y=.fitted)) +
  geom_line(aes(y=.upper),linetype='dotted') +
  geom_line(aes(y=.lower),linetype='dotted') +
  labs(x='AGE',y='PRICE')
```



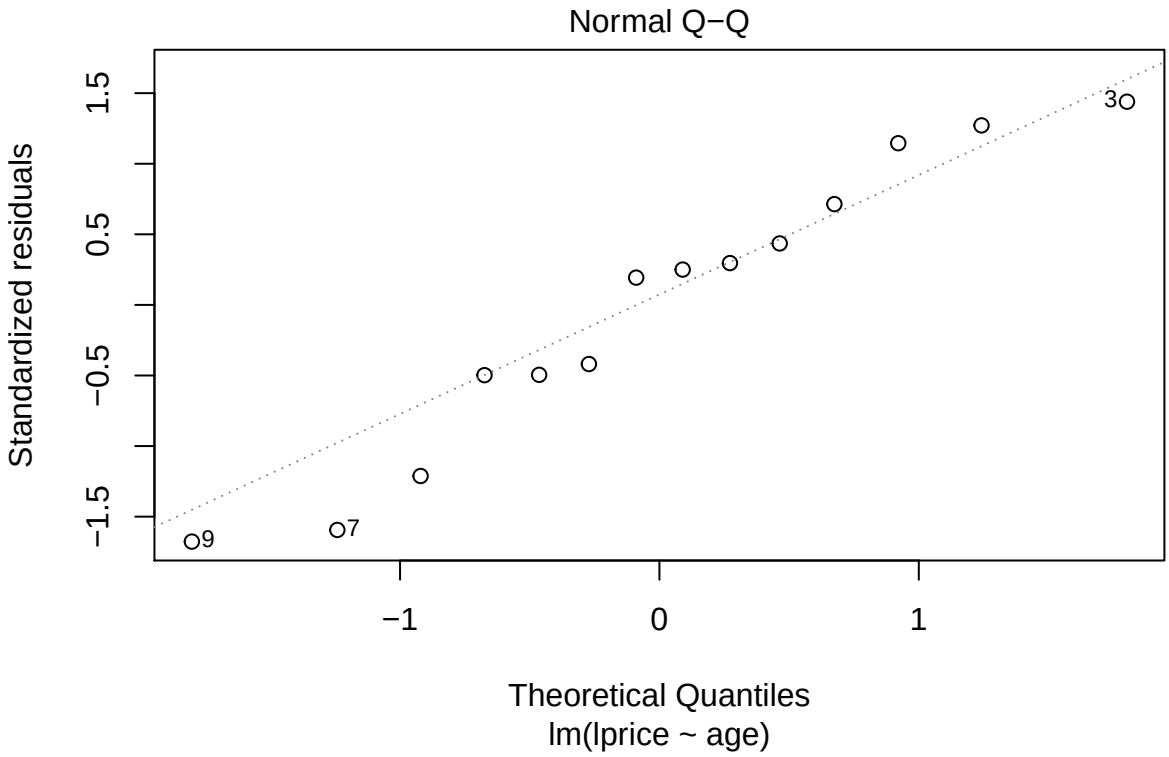
Note that, because the Box-Cox transform is also monotonic, β_1 can still be interpreted as the median effect.

Normalizing Transforms

Transforms aren't just useful for linearizing a relationship, they can also improve the quality of the normality assumptions. For example, we know PRICE>0, and thus in a SLR the error must get truncated. LPRICE has no such restriction, so it resolves the truncation of error.

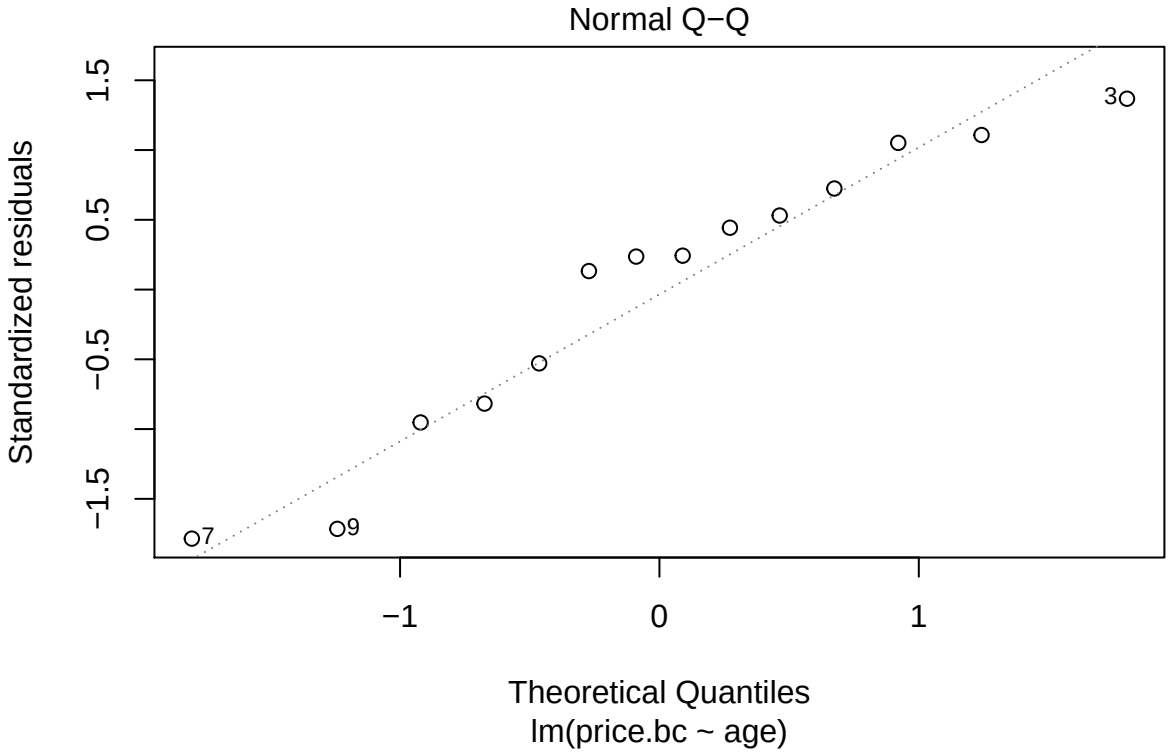
With that said, it doesn't do a great job:

```
plot(mod.log, which=2)
```



The Box-Cox transform, on the other hand, does an excellent job creating normal errors:

```
plot(mod.bc, which=2)
```



In general the Box-Cox transform is recommended for "normalizing" the data.

Constant Variance-izing Transforms

When there is heteroskedasticity (ie. changing variance) we can also use transforms to make the variance more constant.

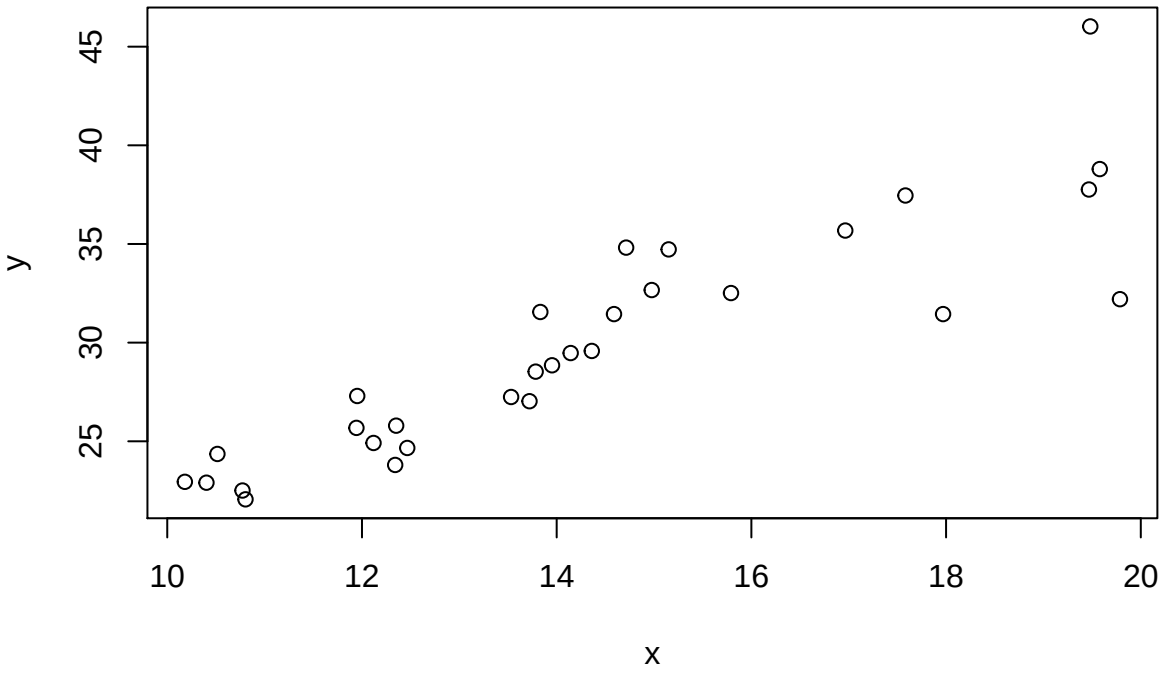
Sometimes, resolving heteroskedasticity is as simple as removing a few outliers. If heteroskedasticity is still present after this, however, you need to identify *which* variable is causing it (eg. using an added-variable plot).

Once you've pinned down which variables are driving heteroskedasticity, we can apply a transform.

Let's look at a toy example:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$
$$\text{Var}[\epsilon_i] = k^2 x_i^2$$

```
set.seed(11235)
x = runif(30,10,20)
k = .1
y = rnorm(x,1+2*x,k^2*x^2)
plot(x,y)
```



In this case, observe that we could rewrite the model

$$\frac{y_i}{x_i} = \frac{\beta_0}{x_i} + \beta_1 + \frac{\epsilon_i}{x_i}$$

And that now, the random error term has variance:

$$\text{Var}[\frac{\epsilon_i}{x_i}] = \frac{\text{Var}[\epsilon_i]}{x_i^2} = k^2$$

This prompts us to choose the transformation:

$$y_i' = \frac{y_i}{x_i}$$
$$x_i' = \frac{1}{x_i}$$
$$\beta_0' = \beta_1$$
$$\beta_1' = \beta_0$$
$$\epsilon' = \frac{\epsilon_i}{x_i}$$

Giving us the new constant-variance model:

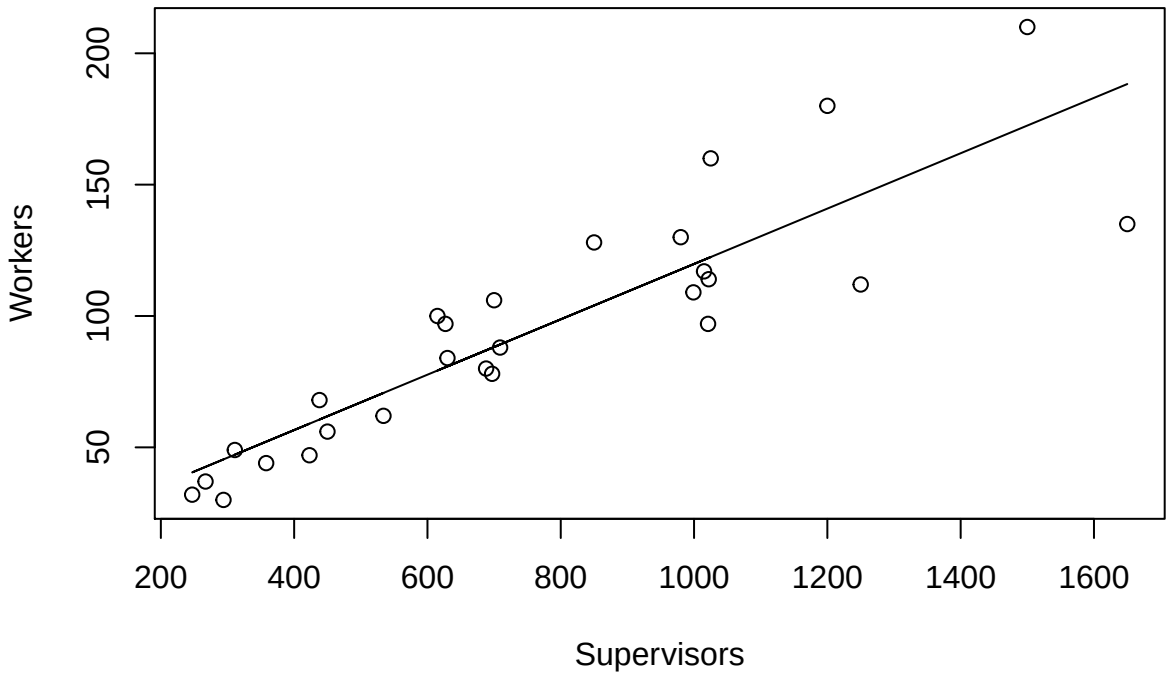
$$y_i' = \beta_0' + \beta_1' x_i' + \epsilon_i'$$

Example: Supervisor Data

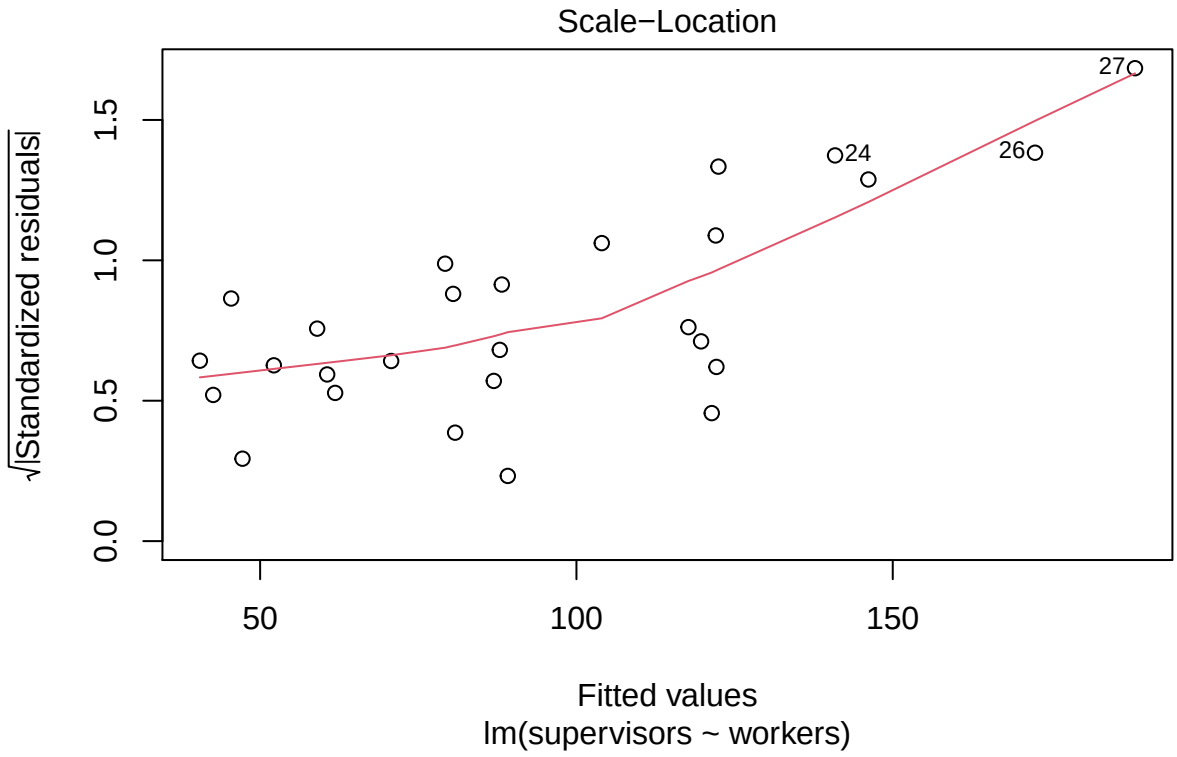
```
supers = read.csv('../../data/supervisor.csv')

supers %>% head
```

```
##   workers supervisors
## 1     294          30
## 2     247          32
## 3     267          37
## 4     358          44
## 5     423          47
## 6     311          49
```

```
mod.supers = lm(supervisors ~ workers, data=supers)

plot(supers$workers,supers$supervisors,xlab='Supervisors',ylab='Workers')
lines(supers$workers, predict(mod.supers))
```



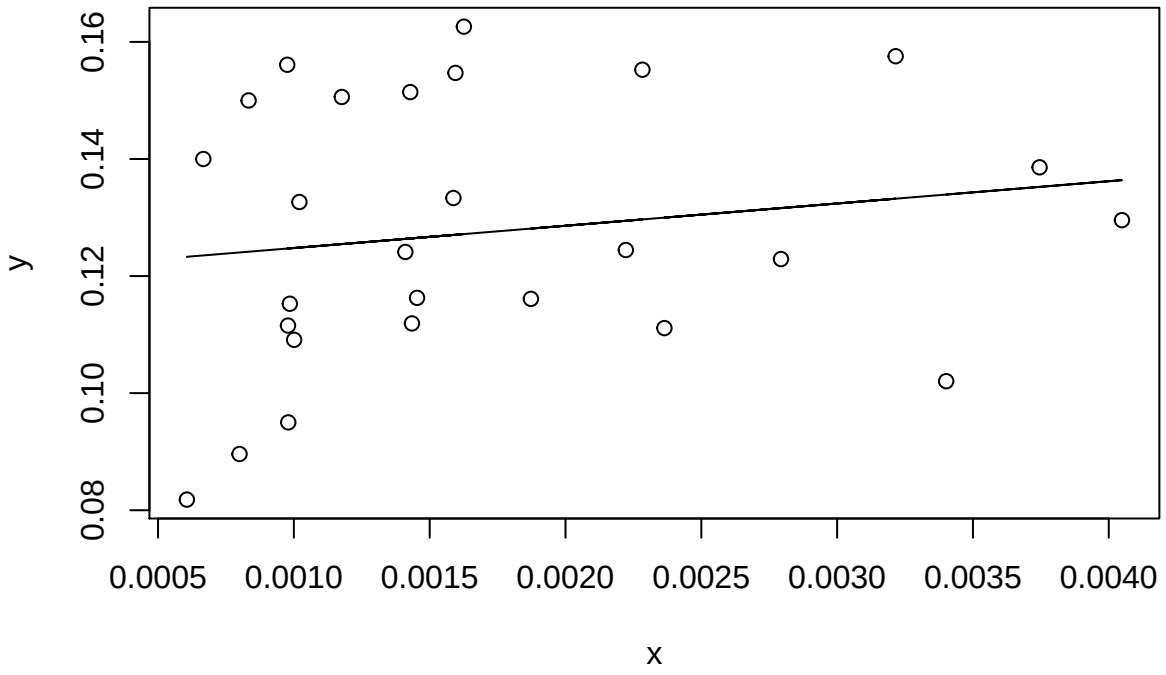
```
mod.supers %>% plot(which=3)
```



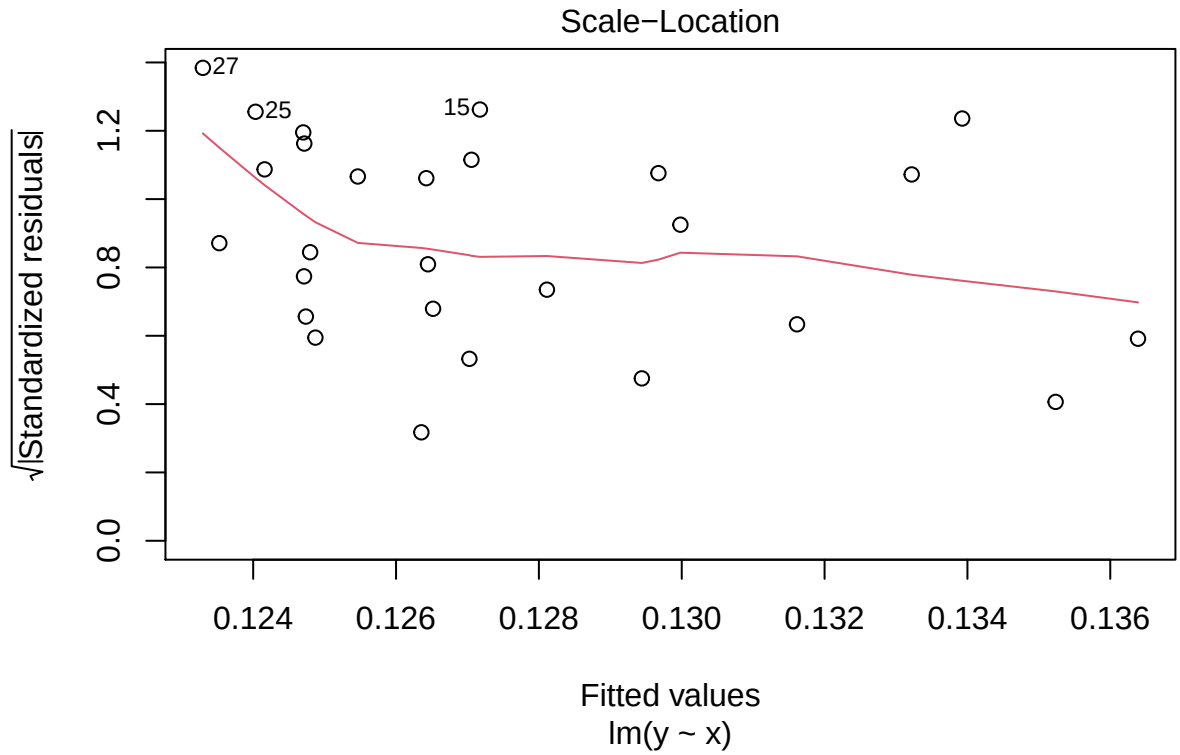
Not that the square root of the studentized residuals is approximately linear, so the variance of the residuals is going as roughly WORKERS^2. Let's apply the transformation from above:

$$x = \frac{1}{\text{WORKERS}} y = \frac{\text{SUPERVISORS}}{\text{WORKERS}}$$

```
x = 1/supers$workers
y = supers$supervisors/supers$workers

mod.supers.transformed = lm(y~x)
plot(x,y)
lines(x,predict(mod.supers.transformed))
```




```
mod.supers.transformed %>% plot(which=3)
```



Weighted Least Squares

Weighted least squares (WLS) is a way to adapt the LSE framework to the case of heteroskedasticity. It's actually quite simple, say that we know:

$$\text{Var}[\epsilon_i] = \sigma_i^2$$

We first define **weights** $w_i = \frac{1}{\sigma_i^2}$.

Then, we choose coefficients $\hat{\beta}_0, ..., \hat{\beta}_p$ by minimizing:

$$WL(\beta_0, ..., \beta_p) = \sum_{i=1}^{n} w_i(y_i - \beta_0 - \beta_1 x_{i1} - ... - \beta_p x_{ip})^2$$

So the weight w_i (and thus the variance $\text{Var}[\epsilon_i]$) determine how heavily the observation $(y_i, \vec{x}_i)$ impact the estimates $\hat{\beta}$.

In the example we just saw, the variance was $\text{Var}[\epsilon_i] \propto x_i^2$, and thus weights are $w_i \propto \frac{1}{x_i^2}$. Note that a constant of proportionality doesn't matter here, since it would be applied to each term in the weighted least squares sum.

Example: Supervisor Data

```
mod.supers.wls = lm(supervisors ~ workers, data=supers, weights=1/workers^2)
mod.supers.wls %>% summary # the WLS model is already on the original scale!
```

```
##
## Call:
## lm(formula = supervisors ~ workers, data = supers, weights = 1/workers^2)
##
## Weighted Residuals:
##       Min        1Q    Median        3Q       Max
## -0.041477 -0.013852 -0.004998  0.024671  0.035427
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.803296   4.569745   0.832    0.413
## workers     0.120990   0.008999  13.445 6.04e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02266 on 25 degrees of freedom
## Multiple R-squared:  0.8785, Adjusted R-squared:  0.8737
## F-statistic: 180.8 on 1 and 25 DF,  p-value: 6.044e-13
```

```
mod.supers.transformed %>% summary
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.041477 -0.013852 -0.004998  0.024671  0.035427
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.120990   0.008999  13.445 6.04e-13 ***
## x           3.803296   4.569745   0.832    0.413
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.02266 on 25 degrees of freedom
## Multiple R-squared:  0.02696,    Adjusted R-squared:  -0.01196
## F-statistic: 0.6927 on 1 and 25 DF,  p-value: 0.4131
```

The residuals produced by WLS are the same as the transformed data:

```
plot(rstudent(mod.supers.wls),rstudent(mod.supers.transformed))
```