Transformations

SDS 291

When regression assumptions aren't met, especially for non-linearity, you can often transform or make changes to your data to address these issues.

Does the number of hospitals reflect the number of available doctors?

```
## Registered S3 method overwritten by 'rvest':
##   method            from
##   read_xml.response xml2
```

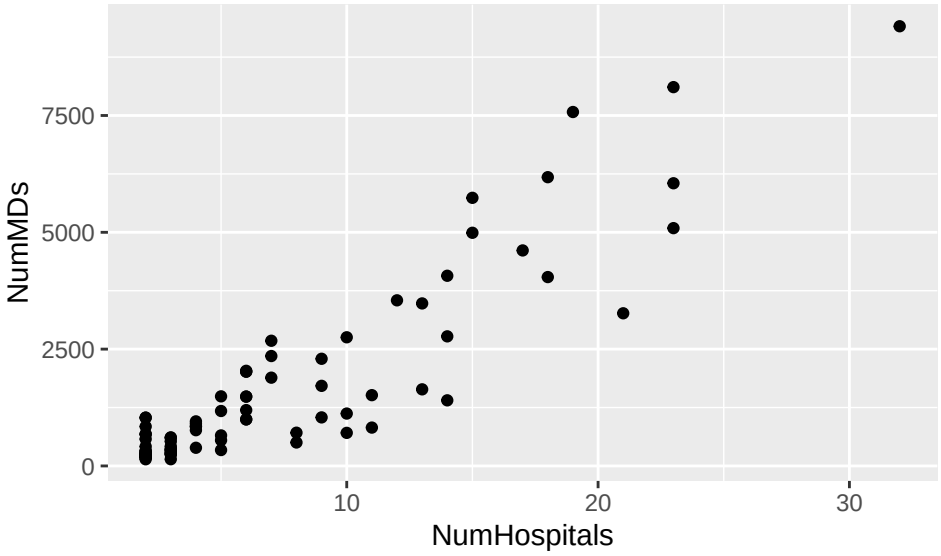
```
## -- Attaching packages --------------------------------------------------------- tidyverse 1.2.1 -
```

```
## v tibble  2.1.3     v purrr   0.3.3
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------ tidyverse_conflicts() -
## x mosaic::count() masks dplyr::count()
## x purrr::cross()  masks mosaic::cross()
## x mosaic::do()    masks dplyr::do()
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x mosaic::stat()  masks ggplot2::stat()
## x mosaic::tally() masks dplyr::tally()
```

Choose

```
qplot(y=NumMDs, x=NumHospitals, data=MetroHealth83)
```



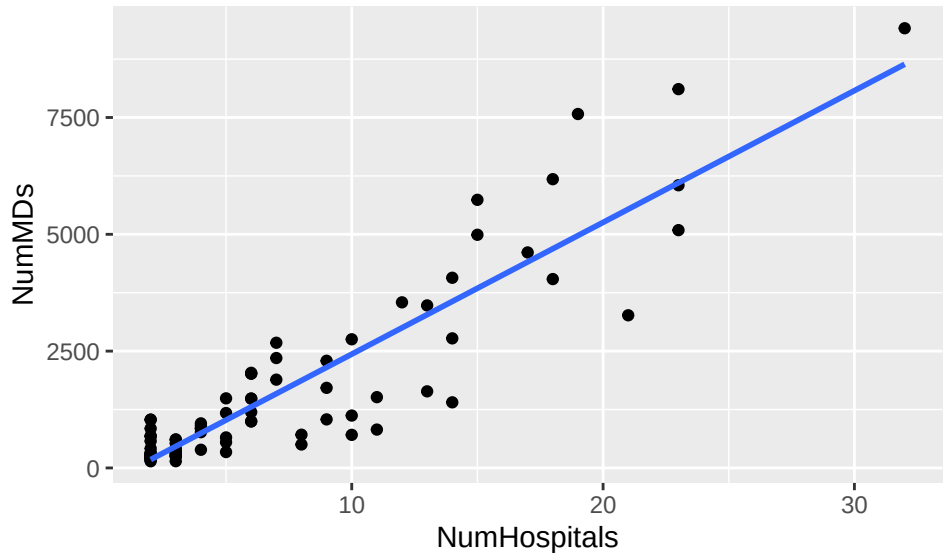
Model we choose to fit:

Fit

```
m2<-lm(NumMDs~NumHospitals, data=MetroHealth83)
summary(m2)
```

```
##
## Call:
## lm(formula = NumMDs ~ NumHospitals, data = MetroHealth83)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2270.09  -263.44    58.08   309.02  2601.93
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -385.10     138.26  -2.785  0.00666 **
## NumHospitals   282.01      14.42  19.563  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 833.2 on 81 degrees of freedom
## Multiple R-squared:  0.8253, Adjusted R-squared:  0.8232
## F-statistic: 382.7 on 1 and 81 DF,  p-value: < 2.2e-16
```

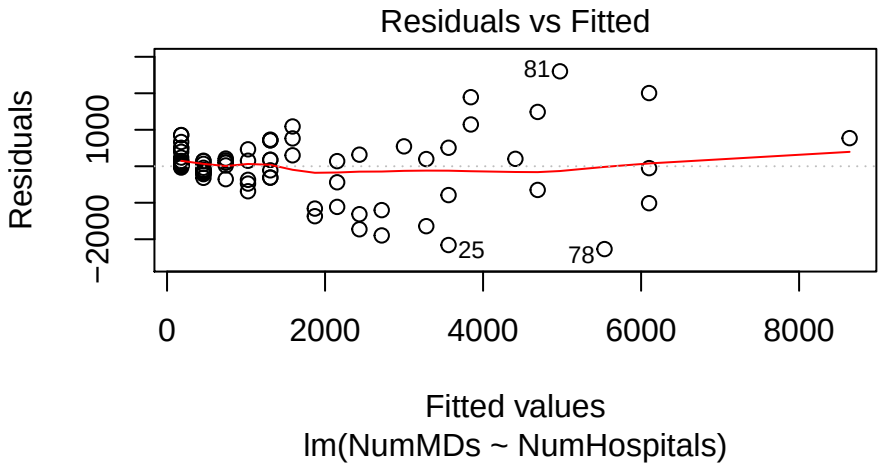
```
qplot(y=NumMDs, x=NumHospitals, data=MetroHealth83) + geom_smooth(method="lm", se=FALSE)
```



Interpretation of findings:

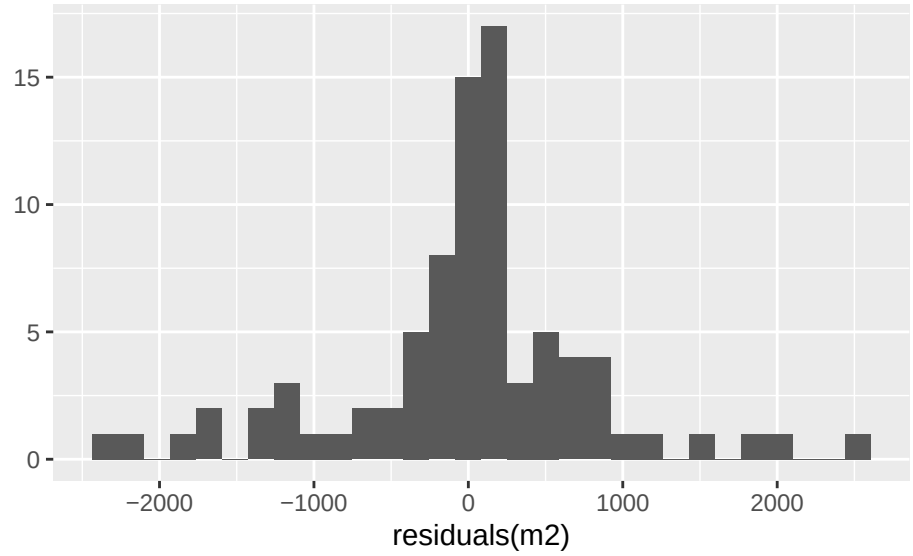
Assess

```
plot(m2, which=1)
```

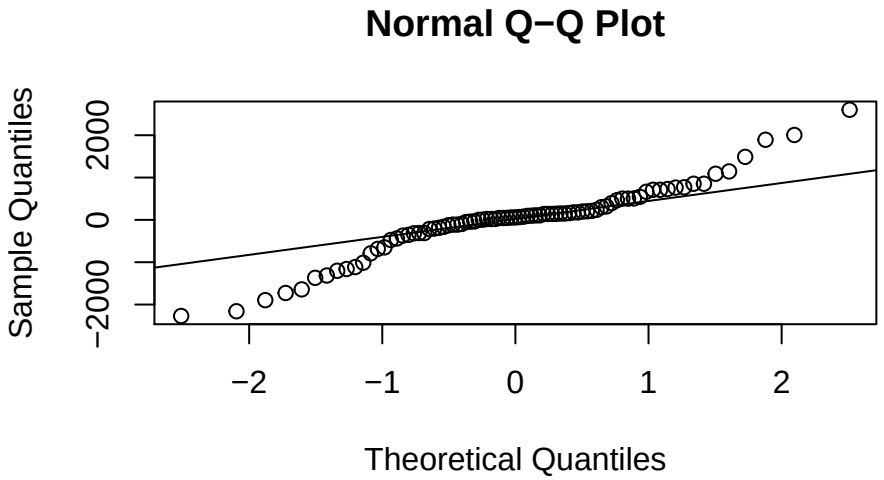



```
qplot(x=residuals(m2), data=m2)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
qqnorm(residuals(m2))
qqline(residuals(m2))
```

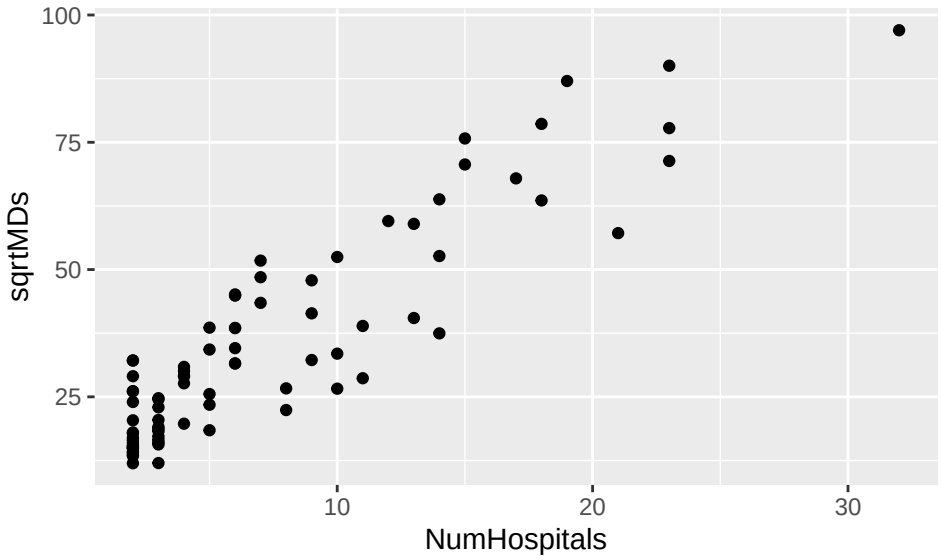


Choose (Again)

From Tukey's Ladder, we can know that a relationship with this kind of shape might be addressed by going "down" in Y, either a square root of Y or a log of Y.

In this case, we'll use the square root.

```
MetroHealth83 = MetroHealth83 %>%
  mutate(sqrtMDs = sqrt(NumMDs))
#MetroHealth83$sqrtMDs<-sqrtMDs(NumMDs)
qplot(y=sqrtMDs, x=NumHospitals, data=MetroHealth83)
```



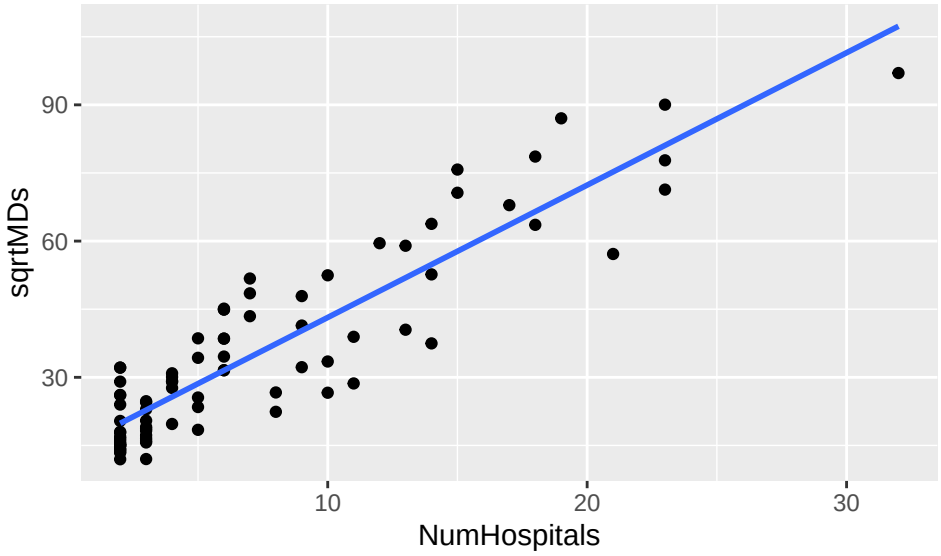
What Model are we choosing?

##Fit (Again)

```
m3<-lm(sqrtMDs~NumHospitals, data=MetroHealth83)
summary(m3)
```

```
##
## Call:
## lm(formula = sqrtMDs ~ NumHospitals, data = MetroHealth83)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -18.086  -5.845  -2.030   7.001  17.994
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   14.0329     1.4686   9.555 6.36e-15 ***
## NumHospitals   2.9148     0.1531  19.036  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.85 on 81 degrees of freedom
## Multiple R-squared:  0.8173, Adjusted R-squared:  0.8151
## F-statistic: 362.4 on 1 and 81 DF,  p-value: < 2.2e-16
```

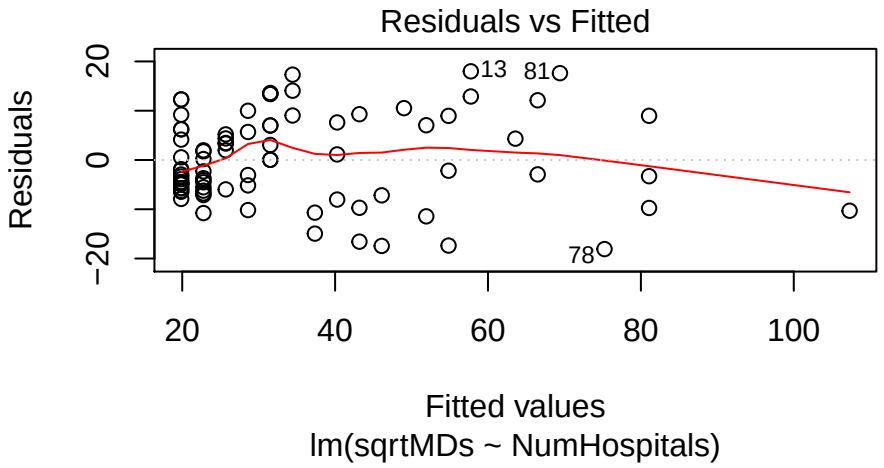
```
qplot(y=sqrtMDs, x=NumHospitals, data=MetroHealth83) + geom_smooth(method="lm", se=FALSE)
```



Interpretation of findings:

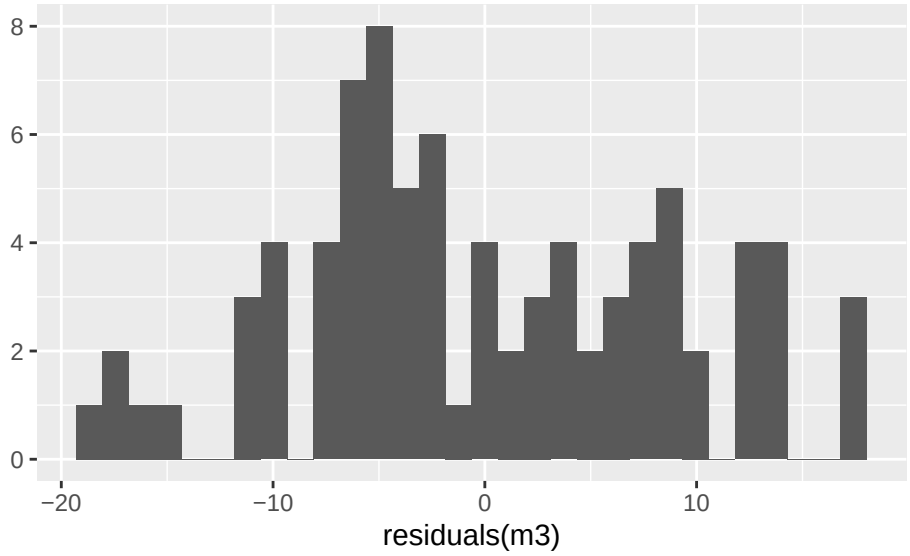
Assess (Again)

```
plot(m3, which=1)
```

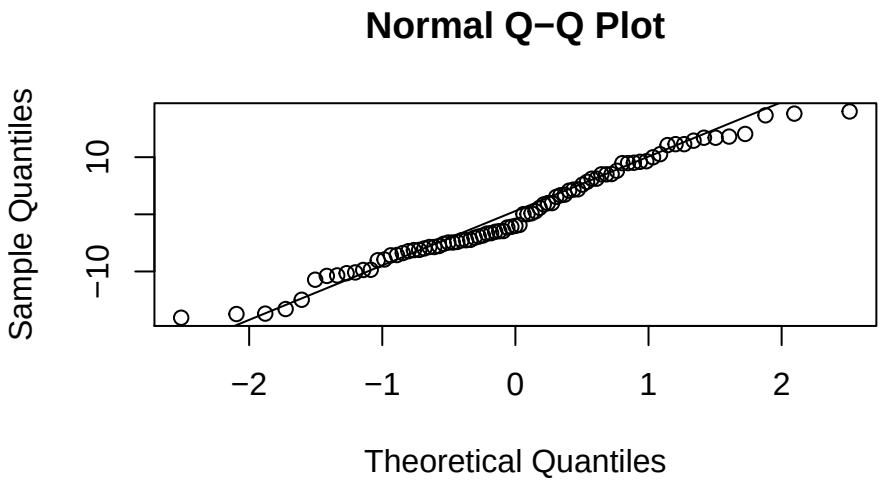



```
qplot(x=residuals(m3))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
qqnorm(residuals(m3))
qqline(residuals(m3))
```



Use (Again)

What if we predict for a city like Louisville that has 18 hospitals how many doctors there would be. Be sure you have the units right!

```
#This is a part of the mosaic package that makes prediction somewhat easier
library(mosaicCore)
predictedMDs<-makeFun(m3)
predictedMDs(NumHospitals=18)
```

```
##        1
## 66.49963
```

```
predictedMDs(NumHospitals=18)^2
```

```
##        1
## 4422.201
```

Extra practice

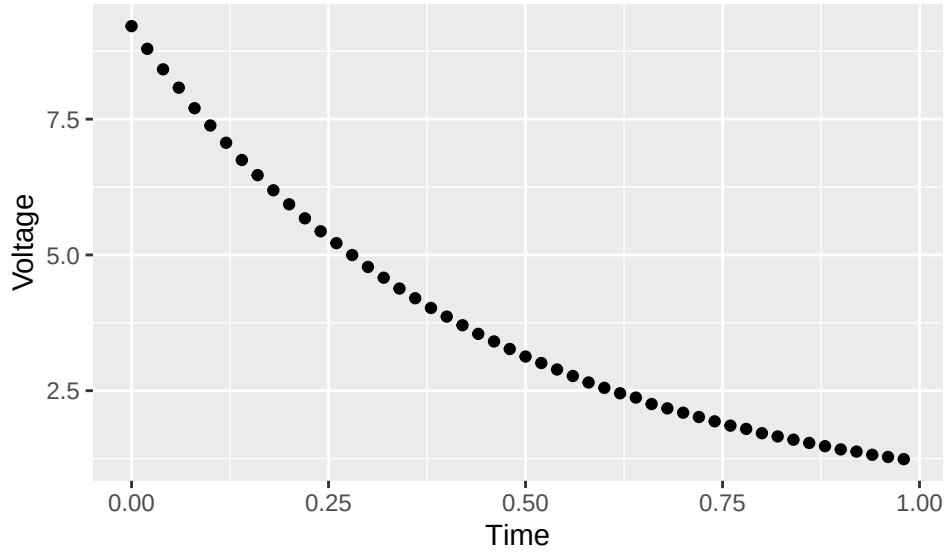
Remember that Tukey's Ladder suggests something with a given shape could be addressed by either changes to Y (those suggested to the right of the red curved line in Tukey's bulge graphic), X (those suggested at the top of the red curved line), or both.

Practice transforming X instead (use code from the SAT example on the slides to help). Do you get the same sort of answer or as good of a fitting model?

Capacitor voltage

A capacitor was chard with a 9-volt battery and then a voltmeter recorded the coltage as the capacitor was discharged. Measurements were taken every 0.02 secton. The data are in the dataset `Volts`

```
data("Volts")
qplot(y=Voltage, x=Time, data=Volts)
```

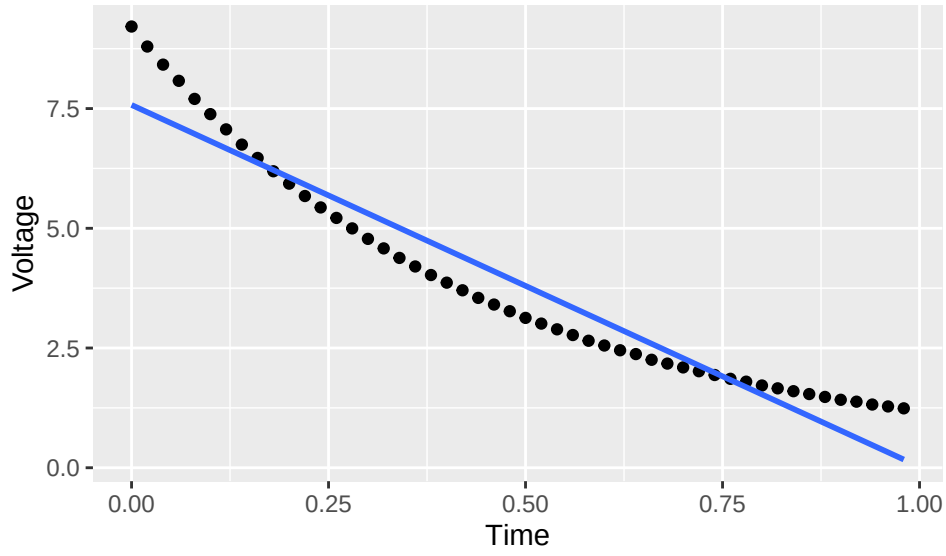



```
m_volts_linear<-lm(Voltage~Time, data=Volts)
summary(m_volts_linear)
```

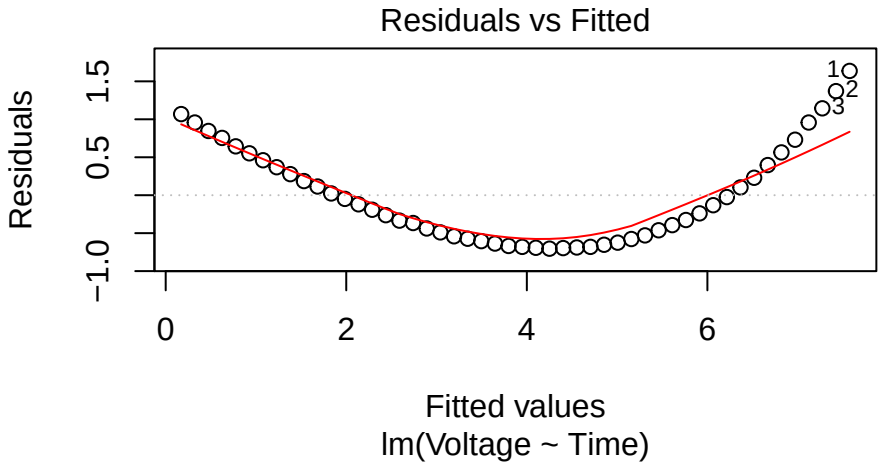
```
##
## Call:
## lm(formula = Voltage ~ Time, data = Volts)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -0.7046 -0.5655 -0.1618  0.4446  1.6375
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   7.5753     0.1793   42.24   <2e-16 ***
## Time         -7.5549     0.3154  -23.95   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6436 on 48 degrees of freedom
## Multiple R-squared:  0.9228, Adjusted R-squared:  0.9212
## F-statistic: 573.9 on 1 and 48 DF,  p-value: < 2.2e-16
```

```
qplot(y=Voltage, x=Time, data=Volts) + geom_smooth(method = "lm", se=FALSE)
```




```
plot(m_volts_linear, which=1)
```



We can see this linear

model is not a good fit for these data – even though there is a small p-value and a big R^2, etc.!

We could interpret this model as saying every 1 second after being charged, the capacitor's voltage decreases by -7.5549 volts.

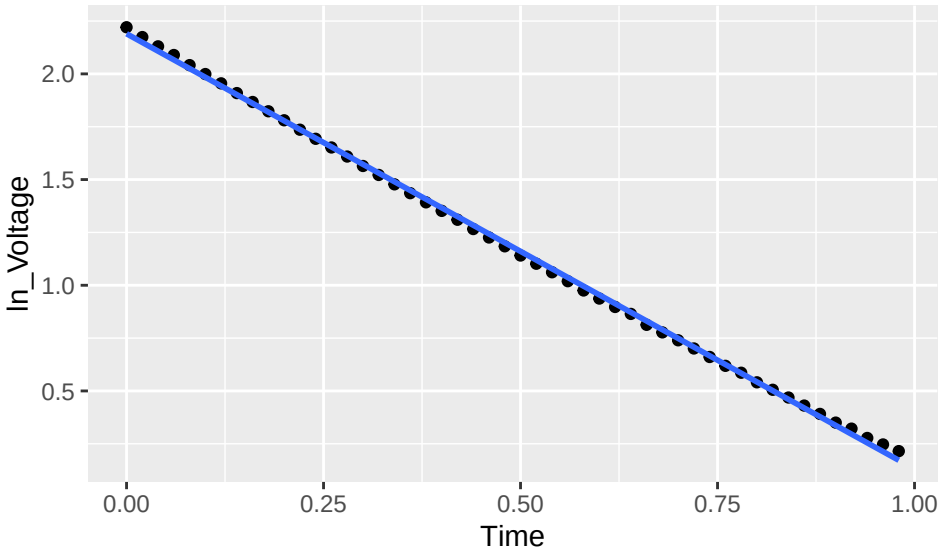
This looks so clearly non-linear. Specifically, this pattern looks like the downward curved shape from Tukey's Ladder, such that we could also transform `Y` with a square root or a log.

In this case, let's use a log.

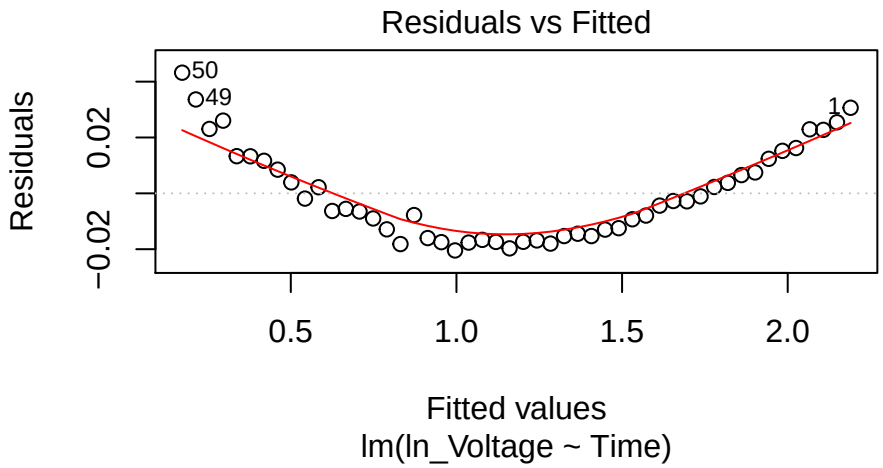
```
Volts <- Volts %>%
  mutate(ln_Voltage = log(Voltage))

m_volts_logged<-lm(ln_Voltage~Time, data=Volts)
summary(m_volts_logged)
```

```
##
## Call:
## lm(formula = ln_Voltage ~ Time, data = Volts)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.020448 -0.015084 -0.003621  0.012190  0.043212
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.189945   0.004637   472.3   <2e-16 ***
## Time        -2.059065   0.008154  -252.5   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01664 on 48 degrees of freedom
## Multiple R-squared:  0.9992, Adjusted R-squared:  0.9992
## F-statistic: 6.377e+04 on 1 and 48 DF,  p-value: < 2.2e-16
```

```
qplot(y=ln_Voltage, x=Time, data=Volts) + geom_smooth(method = "lm", se=FALSE)
```



```
plot(m_volts_logged, which=1)
```



Here, we see an *extremely* strong relationship between Time and the log of Voltage. We could interpret this model as saying every 1 seconds after being charged, the capacitor's voltage decreases by -2.05 logged volts. You can see this visually - when seconds is 0 is the intercept (2.18) and the slope in seconds has gone down by 2.06 log(Volts) after 1 second.

Let's predict what our model would have suggested for how many volts the capacitor would have had a 0.5 seconds:

```
#This is a part of the mosaic package that makes prediction somewhat easier
library(mosaicCore)
predictedVolts<-makeFun(m_volts_logged)
#Volts are still on the log scale
predictedVolts(Time=0.5)
```

```
##        1
## 1.160412
```

```
#Exponentiate to take the predicted value back to the original Volts scale
exp(predictedVolts(Time=0.5))
```

```
##        1
## 3.191248
```

The code above, from the `mosaic` package makes a prediction function (`makeFun()`) that we're calling `predictedVolts` based on our regression model for a given value of the X variable.

The first value is given in the log(Volts) unit - our model would have predicted that the capacitor would have had 1.16 log(Volts) at 0.5 seconds. If you look at the graph with Volts on the log scale above, that looks about right.

But we want *actual* volts. So we can exponentiate that predicted value (hence the `exp()` around the prediction function) and get the actual units (Volts), depicted in the first set of graphs. Thus, we see that

our model - estimated on the log(Volts) scale - would have predicted the capacitor to have had 3.19 Volts at 0.5 seconds. If you look back to the graph on the original scale, that corresponds closely with the actual value at 0.5 seconds (and should, since this model estimated the relationship so closely).

For comparison, let's see what the linear model would have predicted:

```
predictedVolts_linear<-makeFun(m_volts_linear)

predictedVolts_linear(Time=0.5)
```

```
##        1
## 3.797807
```

And the linear model *did* overestimate this relationship, for this part of the model - it was trying to fit something linear to a curved relationship, so it was overestimating the middle times and underestimating the first and last times.

This suggests the log-transformed model is doing a much better job at fitting the relationship between time and voltage.