Lab 4

Week 4 Model Assumptions

Simran Johal

February 2, 2023

Contents

```
# Load required packages

library(foreign) # reading spss files
library(ggplot2) # plotting
library(GGally)
library(ggpubr)
library(car)
library(dplyr) # data wrangling
library(skedastic)
```

We will use the `phobia` and Harvard Alcohol Study data sets to illustrate concepts that were introduced in lecture this week. See the previous labs for more information on these data sets.

```
phobia <- read.spss("phobiasocial.sav", to.data.frame = TRUE)
phobia <- phobia[which(complete.cases(phobia)), ] # To avoid problems later we will remove the case wit

harvard <- read.csv("harvard_alc.csv") |>
  filter(!is.na(DrinkingProblem99) & !is.na(NoDrinksMonth99))
```

As a reminder, `phobia` contains the following variables:

- `SPAI_SP` is the total score on the Social Phobia and Anxiety Inventory: Sobial Phobia Subscale
- `SAS_FNE` is the total score on the Social Anxiety Scale for Adolescents: Fear of worries of negative evaluations from peers subscale

- `SAS_N` is the total score on the Social Anxiety Scale for Adolescents: Social avoidance and distress in new social situations subscale
- `SAS_G` is the total score on the Social Anxiety Scale for Adolescents: Social avoidance and distress generalized social inhibition subscale
- `FNE` is the total score on the Fear of Negative Evaluation Scale
- `SAD` is the total score on the Social Avoidance and Distress Scale
- `ADIS_IV` is the Anxiety Disorders Interview Schedule for DSM-IV (coded as 0 if no social phobia and 1 if social phobic)

`harvard` is a survey of Harvard college students concerning alcohol consumption.

Linear Regression Assumptions

In class this week, we covered the assumptions covering linear regression. In case you've forgotten them, here they are again

1. Independence of observations
    - Usually inferred based on how the study was designed and how data was collected. If observations were not paired, then we usually assume independence of observations.
2. Linearity
    - The outcome is a linear function of the regression parameters, e.g., $y = \beta_0 + \beta_1 x$ and not $y = \exp(\beta_0 + \beta_1 x)$
3. Homoscedasticity
    - The variance **of the residuals** is constant across all values of the predictors
4. Normality
    - **The residuals** are normally distributed across all values of the predictors

Illustrative Example

Let's say we have some data on an outcome y and a predictor x on n people, and that we know the exact data-generating mechanism is as follows

$$\begin{aligned} y_i &= \beta_0 + \beta_1 x_i + \epsilon_i \quad i = 1, 2, \ldots, n \\ \epsilon_i &\sim \text{Normal}(0, \sigma^2) \end{aligned}$$

In words, we can say that the ith observation for y is generated by summing 3 quantities:

1. β_0 (i.e., the intercept)
2. $\beta_1 x_i$ (i.e., the regression coefficient for x multiplied by the ith value of x)
3. ϵ_i (i.e., the residual for observation i)
    - The ith residual is assumed to be a random sample from a normal distribution with variance σ^2.

This form of linear regression should look familiar to you by now, but there is an alternative way of saying the exact same thing

$$y_i \sim \text{Normal}(\beta_0 + \beta_1 x_i, \sigma^2)$$

In words, we can say that the ith data point comes from a normal distribution with mean equal to $\beta_0 + \beta_1 x_i$ and variance σ^2.

Let's assume we know all values in advance so that

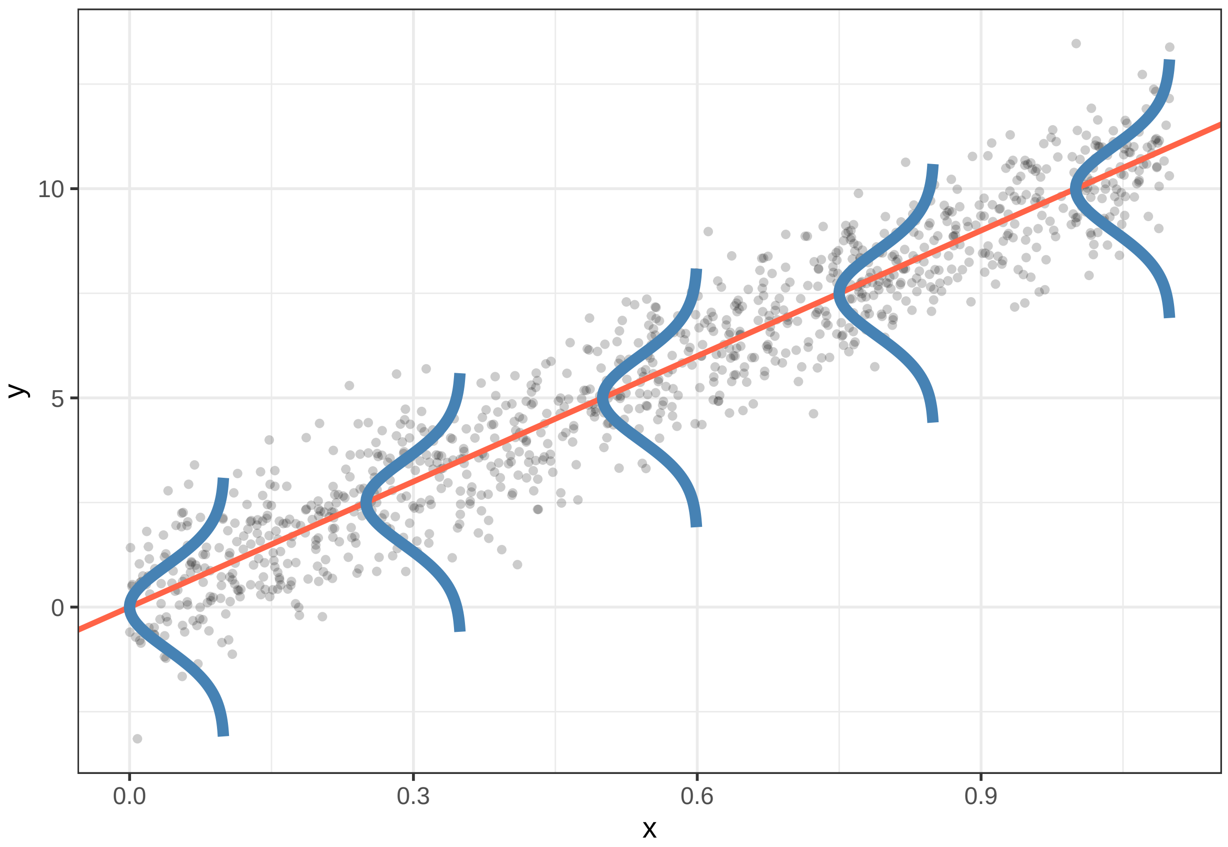
$$\begin{aligned} n &= 1000 \\ \beta_0 &= 0 \\ \beta_1 &= 10 \\ \sigma^2 &= 1 \end{aligned}$$

We can generate this data in R as follows

```r
# set sample size
n <- 1000

# simulate values for our predictor x
x <- runif(n, min = 0, max = 1.1)

# set our values for intercept, slope, and standard deviation
b0 <- 0
b1 <- 10
sigma <- 1

# generate y as we did up above
y <- b0 + b1*x + rnorm(n, mean = 0, sd = sigma)

df <- data.frame(x, y)
```

Below, I've plotted this data with normal distributions superimposed so that we can see the assumptions a bit better, but the code is not necessary to understand for this class.

```r
# The below code can be completely ignored.
curves <- data.frame(x = seq(0, 1, length.out = 5)) |>
     mutate(y_mean = b0 + (b1 * x)) |>
     mutate(ll = qnorm(0.001, mean = y_mean, sd = sigma),
            ul = qnorm(0.999, mean = y_mean, sd = sigma)) |>
     mutate(y = purrr::map2(ll, ul, seq, length.out = 100)) |>
     tidyr::unnest(y) |>
     mutate(density = purrr::map2_dbl(y, y_mean, dnorm,
          sd = sigma)) |>
     mutate(x = x - density * 0.1 / max(density)+0.1)

ggplot(df, aes(x, y)) +
  geom_point(alpha = 0.2, size = 1) +
  geom_abline(intercept = b0, slope = b1, linewidth = 1, col = "tomato") +
  geom_path(data = curves, aes(group = y_mean),
  linewidth = 2, color = "steelblue") +
  theme_bw()
```



Informal Assessments

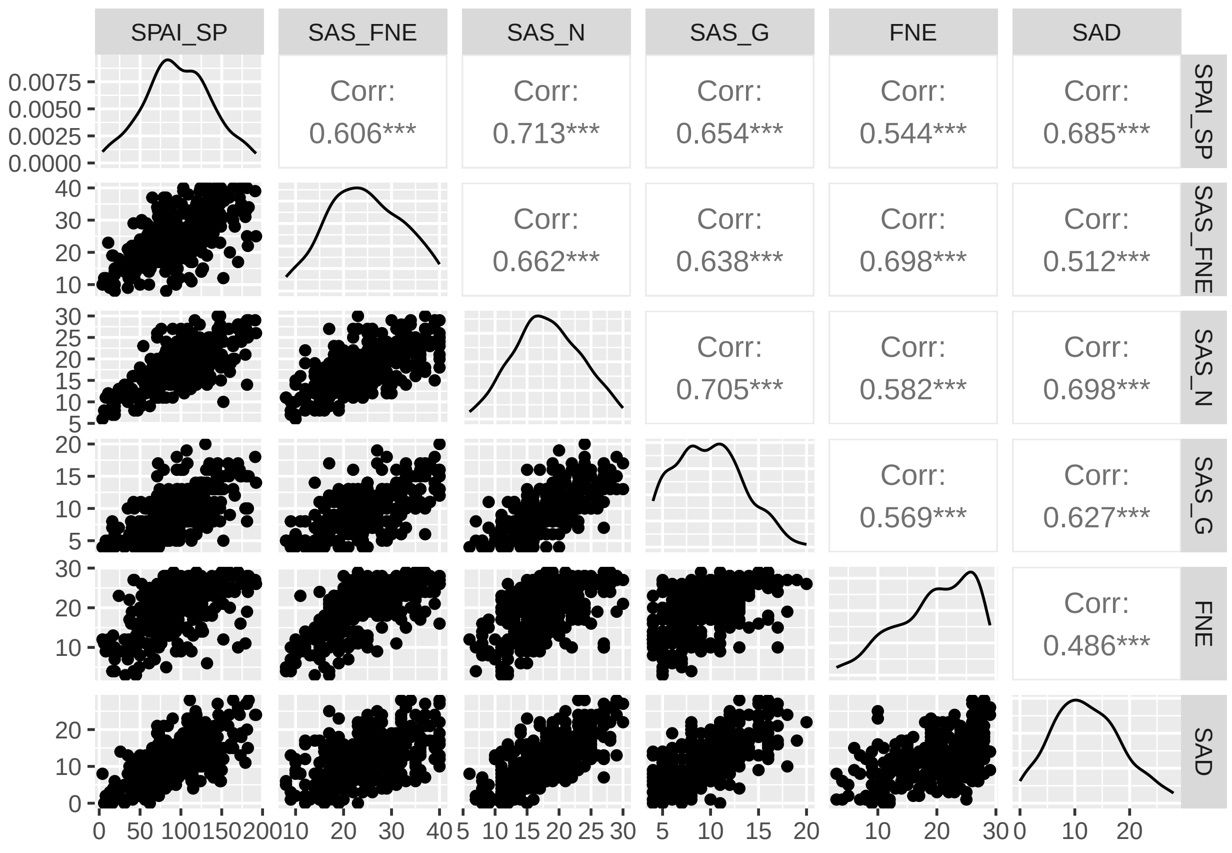
In this section we will review how to evaluate whether assumptions are met and some ways we can try to work around violations.

Let's start by looking at how the continuous variables are associated in the phobia data. The `ggpairs()` function from the `GGally` package allows us to see all pairs of scatterplots, and also provides a density plot for each of the variables and reports the correlations.

This gives us a sense of pairwise linearity, strength of linear associations, and the distribution of the variables in our model.

Let's start by looking at the relationship between the continuous variables in the phobia data.

```
ggpairs(data = phobia,
        columns = 1:6)
```



Based on the output, it looks like `SAS_G` has a slight positive skew, and `FNE` has a negative skew.

Let's review how to test model assumptions concerning the residuals.

Here we will go through two linear regressions — a "good example" and a "bad example"

```
good_example <- lm(SPAI_SP ~ FNE, data = phobia)

bad_example <- lm(DrinkingProblem99 ~ NoDrinksMonth99, data = harvard)
```

First, let's refresh ourselves on what these models predicted:

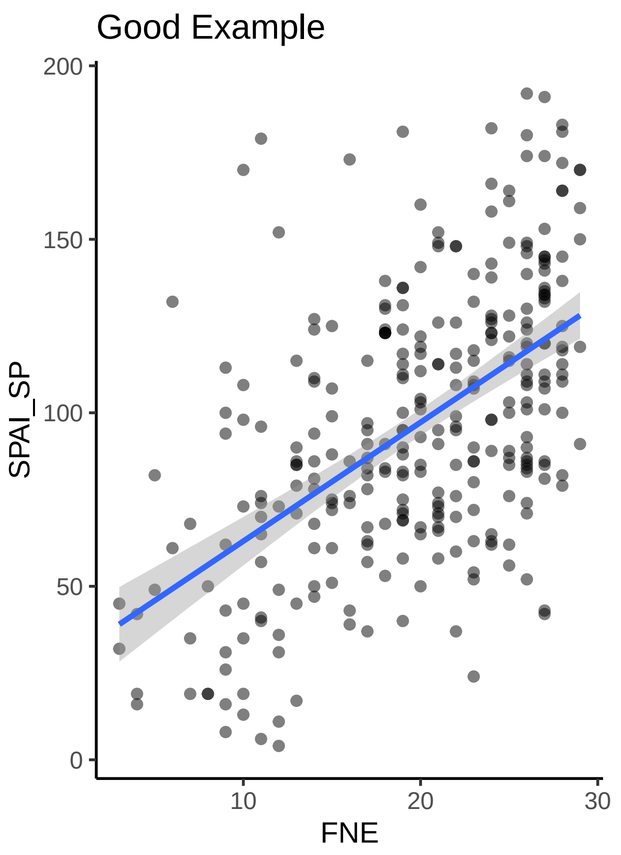
```
summary(good_example)
```

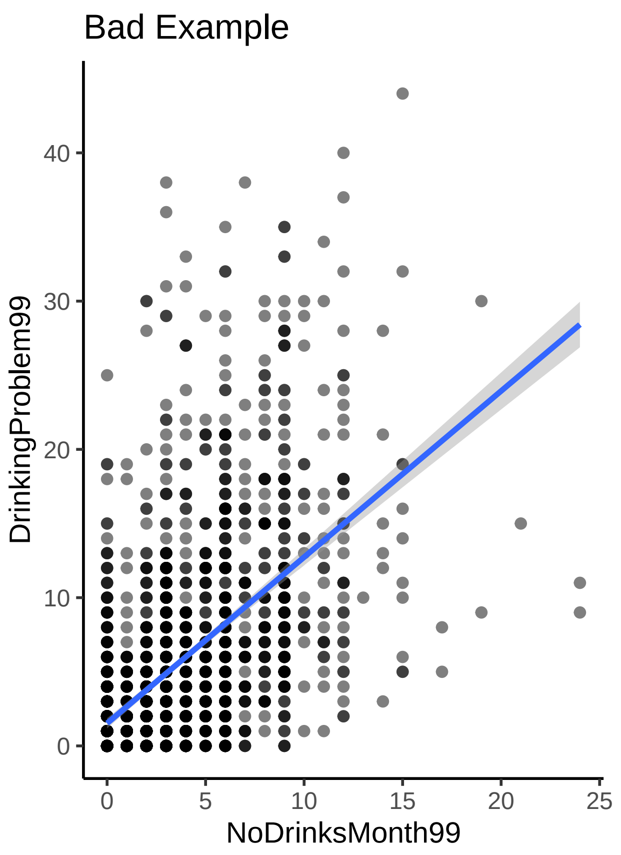
```
##
## Call:
## lm(formula = SPAI_SP ~ FNE, data = phobia)
##
## Residuals:
##    Min      1Q Median      3Q     Max
## -83.52 -25.27  -0.49  21.88 112.57
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  28.7652     6.3271   4.546 7.93e-06 ***
## FNE           3.4241     0.3047  11.236  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 33.73 on 300 degrees of freedom
## Multiple R-squared:  0.2962, Adjusted R-squared:  0.2938
## F-statistic: 126.2 on 1 and 300 DF,  p-value: < 2.2e-16
```

```r
summary(bad_example)
```

```
##
## Call:
## lm(formula = DrinkingProblem99 ~ NoDrinksMonth99, data = harvard)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -19.421  -2.768  -1.527   1.473  33.111
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      1.52731    0.17744   8.607   <2e-16 ***
## NoDrinksMonth99  1.12058    0.03727  30.065   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.7 on 1974 degrees of freedom
## Multiple R-squared:  0.3141, Adjusted R-squared:  0.3137
## F-statistic: 903.9 on 1 and 1974 DF,  p-value: < 2.2e-16
```

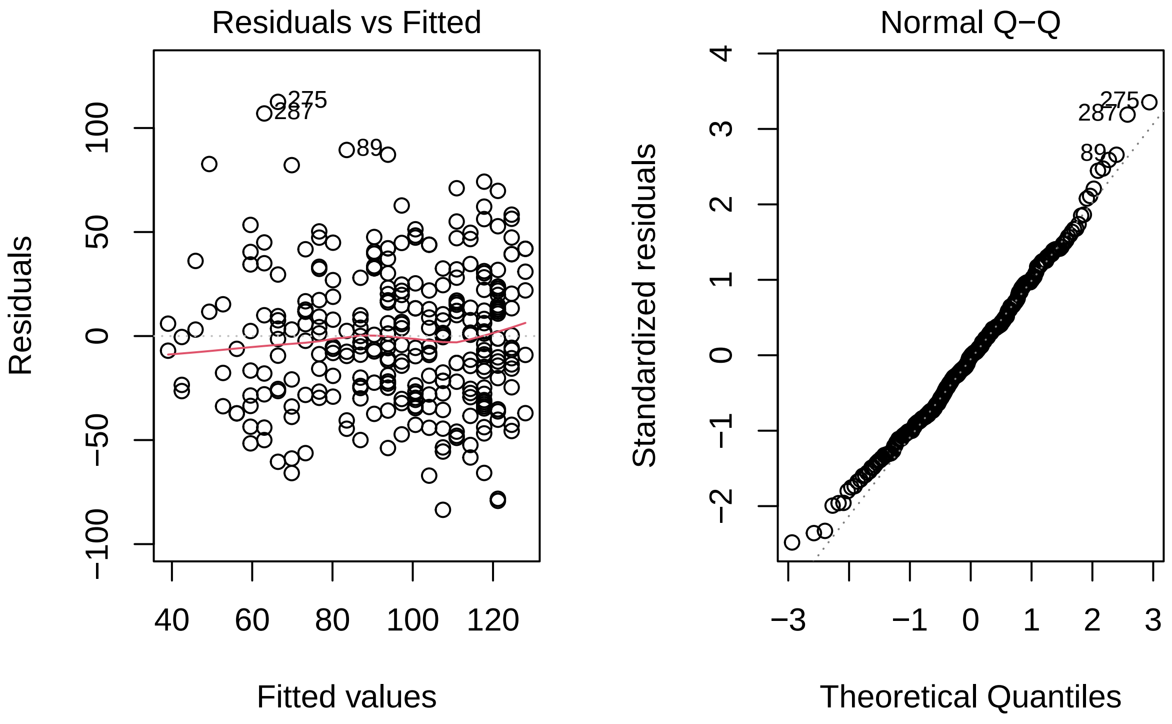
```r
g_good <-
  ggplot(data = phobia,
       aes(x = FNE, y = SPAI_SP)) +
  geom_point(alpha = .5) +
  geom_smooth(method = "lm") +
  labs(title = "Good Example") +
  theme_classic()

g_bad <-
  ggplot(data = harvard,
       aes(x = NoDrinksMonth99, y = DrinkingProblem99)) +
  geom_point(alpha = .5) +
  geom_smooth(method = "lm") +
  labs(title = "Bad Example") +
  theme_classic()

ggarrange(g_good, g_bad, ncol = 2)
```





R has some built-in functionality to visually assess whether we violated the assumptions of linear regression. To use this functionality, we use the `plot()` function and give it the name of our model object. This returns 4 plots. For now, we will only concern ourselves with the first two. Thus, inside of our `plot()` call, we will specify the argument `which`

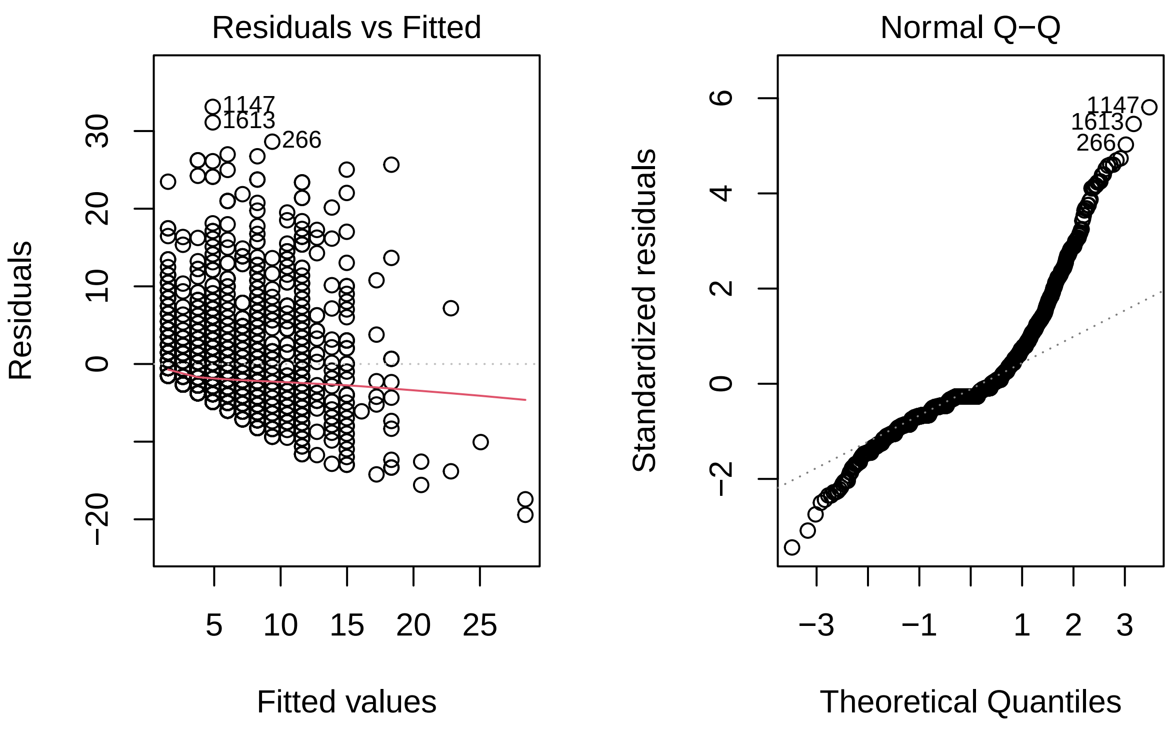
```
# return our plots in 1 row with two columns
par(mfrow = c(1, 2))
plot(good_example, which = c(1,2))
```



Overall, the assumptions look pretty good.

1. Linearity: On the residuals vs fitted graph, the red line is relatively flat around 0, suggesting that there is no relationship between the predicted values and residuals.
2. Homogeneity: On the residuals vs fitted graph, the data are pretty equally scattered, suggesting homogeneity of variance. However, there are more data points on the right than left hand side of the graph.
3. Normality: The Normal Q-Q plot and histogram show that the residuals are distributed relatively normally, although there is a slight positive skew.

```
par(mfrow = c(1, 2))
plot(bad_example, which = c(1,2))
```



1. Linearity: The red line is not very flat at 0, so we might have some problems with the linearity assumption. It seems that at larger predicted scored, the residuals tend to be more negative.
2. Homogeneity: There is a clear problem here, as there is a floor effect and this does not allow an even scattering.
3. Normality: The Q-Q plot indicates the residuals are not normally distributed, with heavy tails. This often happens when you have floor effects with your data.

Formal Assessments

Thus far we have assessed model assumptions by visual inspection, but there are statistical tests to assess our assumptions.

Homogeneity of Variance

White's Test A more formal assessment of the assumption of homogeneity can be done using White's Test. To compute this, we will use the `white()` function from the `skedastic` package.

```
white(good_example)
```

```
## # A tibble: 1 x 5
##   statistic p.value parameter method       alternative
##       <dbl>   <dbl>     <dbl> <chr>        <chr>
## 1     0.958   0.619         2 White's Test greater
```

```
white(bad_example)
```

```
## # A tibble: 1 x 5
##   statistic  p.value parameter method       alternative
##       <dbl>    <dbl>     <dbl> <chr>        <chr>
## 1      163. 3.77e-36         2 White's Test greater
```

In general:

- statistic: This indicates the χ^2 statistic. Larger values indicate more heteroskedasticity.
- p.value: This indicates the likelihood of obtaining the observed χ^2 value in the χ^2 distribution (with the given degrees of freedom, indicated by parameter). If $p < .05$, you have a statistically significant violation of the homogeneity assumption. If $p > .05$, then we can't reject the null hypothesis of homogeneity of variance.

For the good example:

- White's test did not indicate heteroscedasticity $\chi^2(2) = 0.96, p = .62$.

For the bad example:

- White's test indicated there was a significant violation of the homogeneity assumption, $\chi^2(2) = 163.13, p < .001$.

For more information on this test, see: https://en.wikipedia.org/wiki/White_test

Score Test for Non-Constant Error Variance The non-constant error variance test, aka the Breusch–Pagan test, is an alternative to White's Test. To compute this, we will use the `ncvTest` function from the `car` package.

```
ncvTest(good_example)
```

```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 0.5747264, Df = 1, p = 0.44839
```

```
ncvTest(bad_example)
```

```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 573.5607, Df = 1, p = < 2.22e-16
```

In general:

- statistic: This indicates the χ^2 statistic. Larger values indicate more heteroskedasticity.
- p.value: This indicates the likelihood of obtaining the observed χ^2 value in the χ^2 distribution (with the given DF, indicated by parameter). If $p < .05$, you have significant violation of the homogeneity assumption. If $p > .05$, there is not a violation of the homogoeneity assumption.

For the good example:

- A Breusch-Pagan test did not indicate heteroscedasticity of the constant variance assumption, $\chi^2(1) = 0.57, p = .45$.

For the bad example:

- A Breusch-Pagan test indicated there was a significant violation of the homogeneity assumption, $\chi^2(1) = 573.56, p < .001$.

Normality

To test whether the residuals are normally distributed, we can use standard tests for normality on the residuals from the model. Perhaps the most common is the Shapiro-Wilk test.

```
shapiro.test(good_example$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  good_example$residuals
## W = 0.99181, p-value = 0.09302
```

```r
shapiro.test(bad_example$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  bad_example$residuals
## W = 0.84588, p-value < 2.2e-16
```

Good example:

- A Shapiro-Wilk test showed did not provide evidence that the residuals were non-normally distributed, $W = 0.99, p = .09$

Bad example:

- A Shapiro-Wilk test rejected the null hypothesis that the model residuals were normally distributed, $W = 0.85, p < .001$

Transforming Data to Meet Assumptions

Sometimes you can transform your data to meet assumptions.

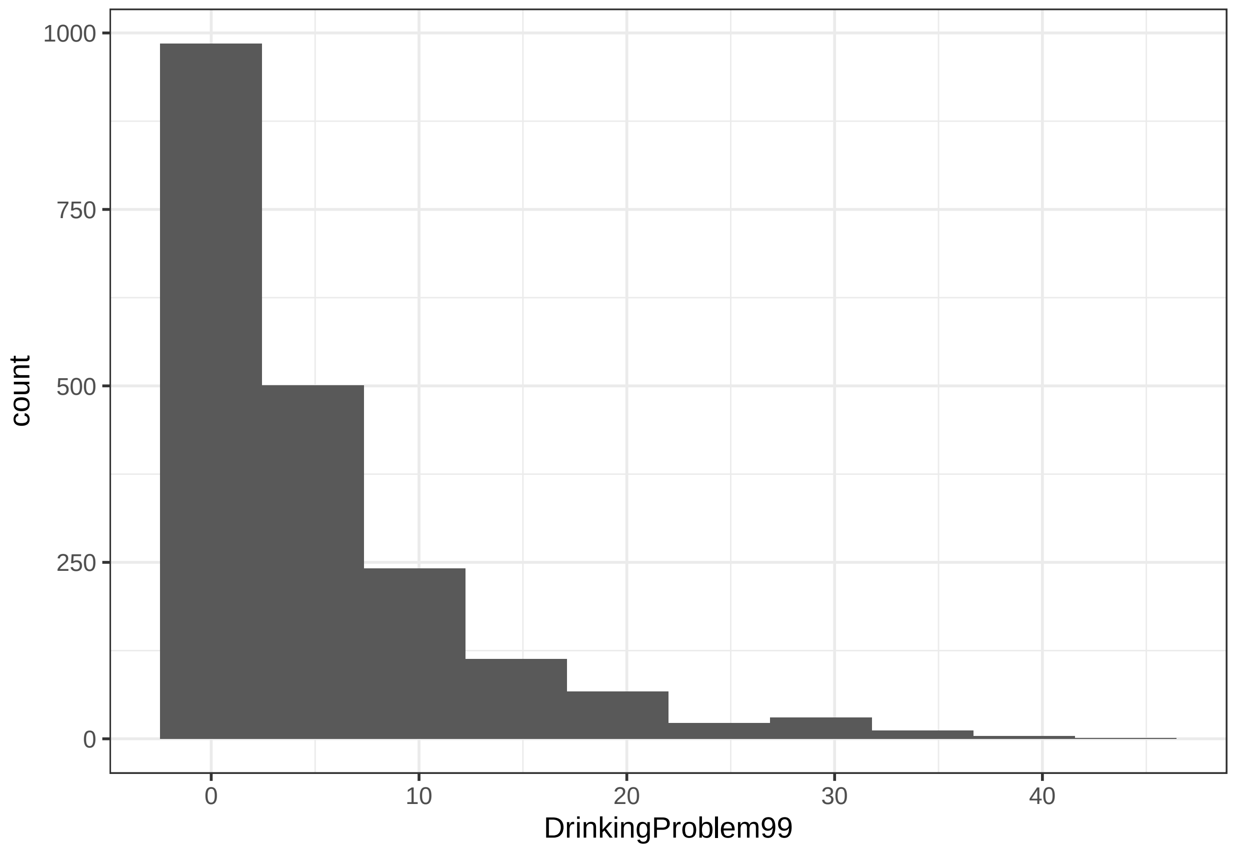
Let's see if we can fix the bad example.

Common transformations include:

- **square** (especially useful for negative skew)
- **square-root**
- **cube-root**
- **logarithm** (especially useful for positive skew)

We can begin by taking a closer look at how the outcome is distributed.

```r
ggplot(harvard, aes(DrinkingProblem99)) +
  geom_histogram(bins = 10) +
  theme_bw()
```

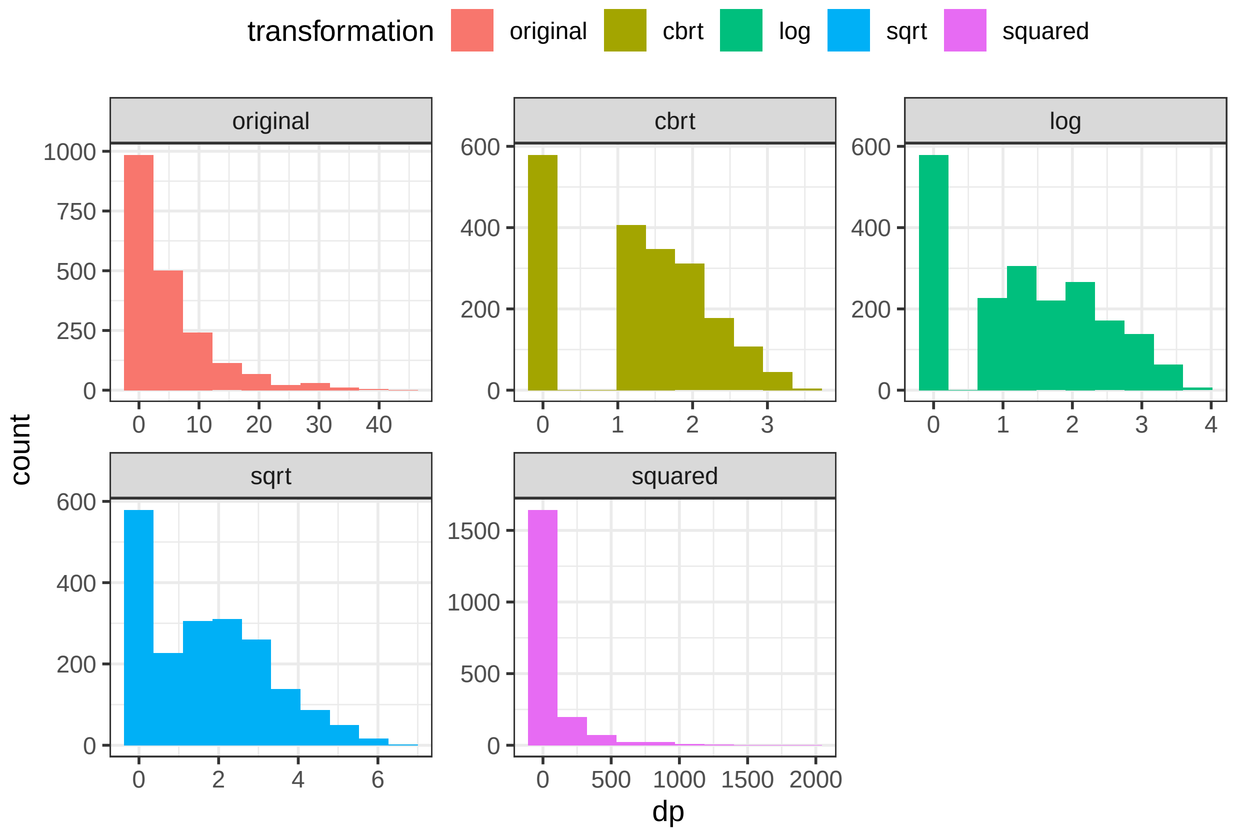


Okay, a pretty skewed distribution. Economists love taking the log of stuff that looks like this (e.g., wages, housing prices, etc.), but for the sake of the lab we will do all of the aforementioned transformations.

A quick note: when you want to take the log of your outcome variable but it contains zero values (in this case, some people reported 0 for `DrinkingProblem99`), we usually add a 1 to all values. This is to avoid the fact that the log of 0 is undefined. Often, the truth is unavoidable, but in this case there is a simple workaround.

```
harvard_transformed <-
  harvard |>
  mutate(dp_squared = DrinkingProblem99^2,
         dp_sqrt = sqrt(DrinkingProblem99),
         dp_cbrt = DrinkingProblem99^(1/3),
         dp_log = log(DrinkingProblem99 + 1))
```

```
harvard_transformed_long <-
  harvard_transformed |>
  rename(dp_original = DrinkingProblem99) |>
  tidyr::pivot_longer(dp_original:dp_log,
                      names_to = "transformation",
                      values_to = "dp",
                      names_prefix = "dp_") |>
  mutate(transformation = factor(transformation),
         transformation = relevel(transformation, ref = "original"))


ggplot(harvard_transformed_long, aes(dp)) +
  geom_histogram(aes(fill = transformation), bins = 10) +
  facet_wrap(~ transformation, scales = "free") +
  theme_bw() +
  theme(legend.position = "top")
```

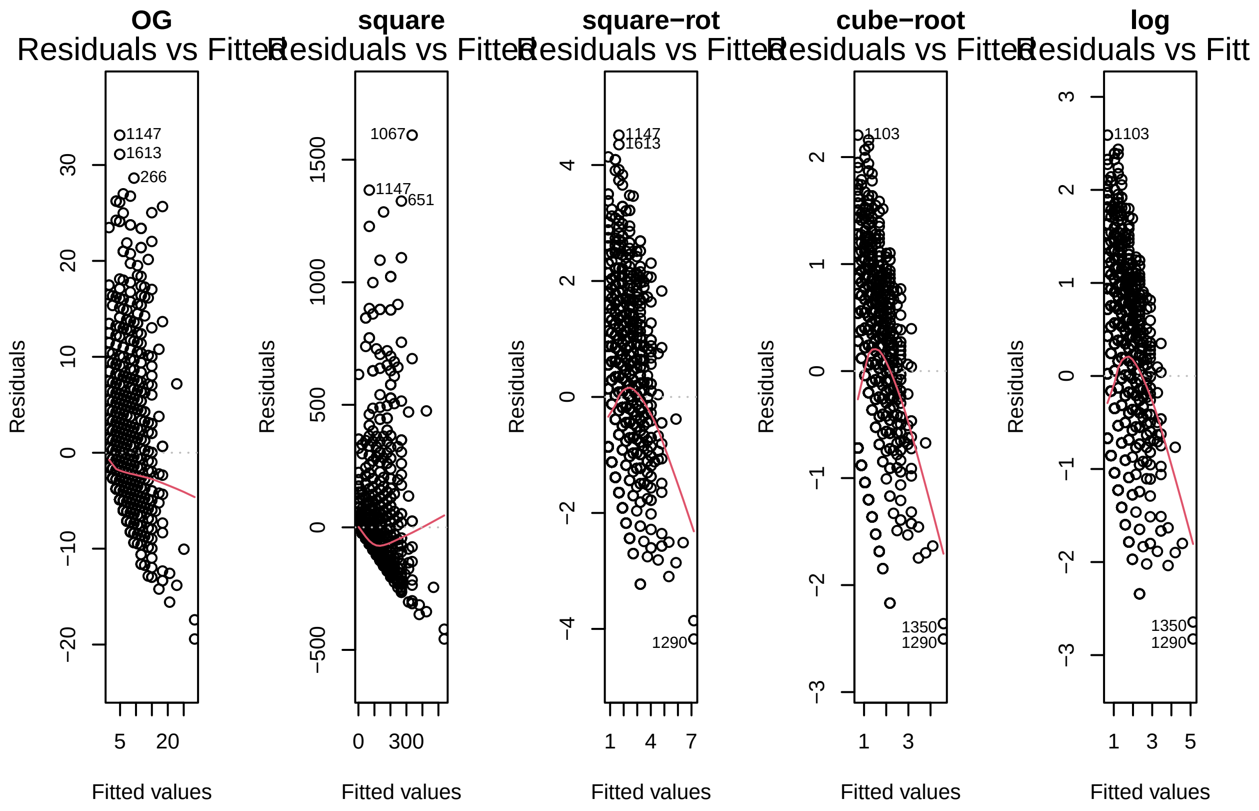


Okay, so none of these are *great*, but perhaps the log does the best job of normalizing our outcome variable here.

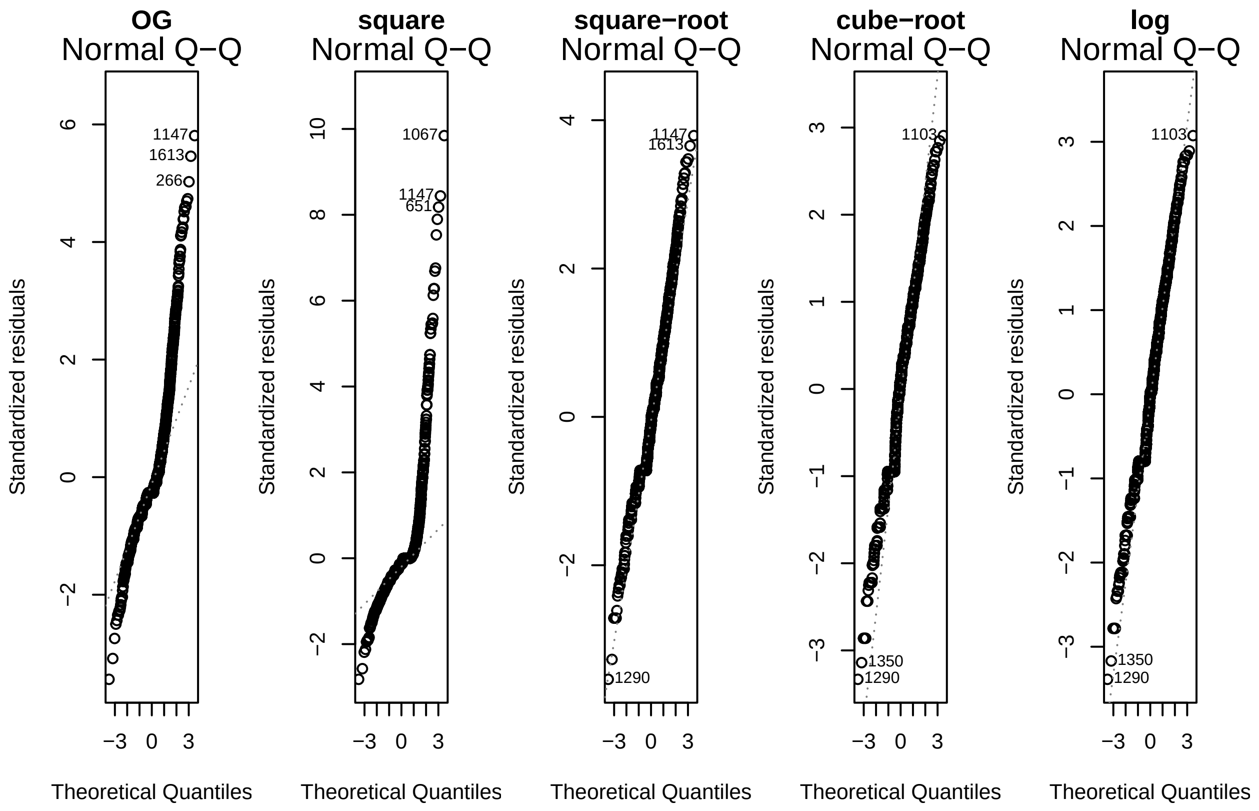
Let's see test them out and see how each of these transformations affect the model assumptions.

```
bad_example_square <- lm(dp_squared ~ NoDrinksMonth99, data = harvard_transformed)
bad_example_sqrt <- lm(dp_sqrt ~ NoDrinksMonth99, data = harvard_transformed)
bad_example_cbrt <- lm(dp_cbrt ~ NoDrinksMonth99, data = harvard_transformed)
bad_example_log <- lm(dp_log ~ NoDrinksMonth99, data = harvard_transformed)
```

```
par(mfrow = c(1, 5))

plot(bad_example, which = 1, main = "OG")
plot(bad_example_square, which = 1, main = "square")
plot(bad_example_sqrt, which = 1, main = "square-rot")
plot(bad_example_cbrt, which = 1, main = "cube-root")
plot(bad_example_log, which = 1, main = "log")
```



```
plot(bad_example, which = 2, main = "OG")
plot(bad_example_square, which = 2, main = "square")
plot(bad_example_sqrt, which = 2, main = "square-root")
plot(bad_example_cbrt, which = 2, main = "cube-root")
plot(bad_example_log, which = 2, main = "log")
```



Based on these graphs it looks like the Square Root or log transformation improves the assumptions the most. But let's test these assumptions more formally.

```r
results <-
  rbind.data.frame(white(bad_example)[1, 1:2],
                   white(bad_example_square)[1, 1:2],
                   white(bad_example_sqrt)[1, 1:2],
                   white(bad_example_cbrt)[1, 1:2],
                   white(bad_example_log)[1, 1:2]) |>
  mutate(model =  c("Untransformed Model",
                    "Square",
                    "Square Root",
                    "Cube Root",
                    "Log"))
results
```

```
## # A tibble: 5 x 3
##   statistic  p.value model
##       <dbl>    <dbl> <chr>
## 1     163.  3.77e-36 Untransformed Model
## 2     118.  2.42e-26 Square
## 3      73.4 1.13e-16 Square Root
## 4      76.7 2.20e-17 Cube Root
## 5      45.1 1.60e-10 Log
```

It looks like the Log transformation and Square Root transformation led to the largest reduction in the test statistic, but there is still a significant violation in all cases.

It looks like the log transformation led to the largest reduction in the test statistic for the Shapiro-Wilk test.

```r
results <-
  rbind.data.frame(shapiro.test(bad_example$residuals),
                   shapiro.test(bad_example_square$residuals),
                   shapiro.test(bad_example_sqrt$residuals),
                   shapiro.test(bad_example_cbrt$residuals),
                   shapiro.test(bad_example_log$residuals))
results |> dplyr::select(-data.name, -method)
```

```
##   statistic      p.value
## 1 0.8458765 3.593781e-40
## 2 0.6106301 5.688104e-55
## 3 0.9723748 4.611661e-19
## 4 0.9784182 1.048365e-16
## 5 0.9786761 1.351954e-16
```

It seems that the log transformation improved the normality assumption the most, but in both cases there is still a significant violation.

Interpreting the results of the transformed models:

Let's say we chose the log transformation to go with.

```r
bad_example
```

```
##
## Call:
## lm(formula = DrinkingProblem99 ~ NoDrinksMonth99, data = harvard)
##
## Coefficients:
##     (Intercept)  NoDrinksMonth99
##           1.527            1.121
```

```r
bad_example_log
```

```
##
## Call:
## lm(formula = dp_log ~ NoDrinksMonth99, data = harvard_transformed)
##
## Coefficients:
##     (Intercept)  NoDrinksMonth99
##          0.6695           0.1858
```

The untransformed model:

$$\widehat{\mathrm{dp}}_i = 1.52 + 1.12\mathrm{ndm}_i$$

- For every one unit change in `NoDrinksMonth99`, there is a predicted increase of 1.12 increase in `DrinkingProblems99`.

The log-transformed model:

$$\widehat{\log(\mathrm{dp}_i)} = 0.67 + 0.19\mathrm{ndm}_i$$

- For every unit change in `NoDrinksMonth99`, there is a predicted increase of 0.19 unit increase in log `DrinkingProblems99`.
- This is the downside of transformed models. They can be difficult to interpret. In this case, an alternative is to exponentiate both sides for a different interpretation, but not necessarily an easier one

$$\widehat{\mathrm{dp}}_i = \exp(0.67 + 1.12\mathrm{ndm}_i)$$

To transform or not to transform?

You should think critically about whether to transform your variables or not. In this example transforming the outcome helped a little, but not much, and this will often be the case in real life.

You should transform your data if there is good evidence that the assumptions are not met with the raw data and that transforming the data will help meet those assumptions. You should also think critically about whether the independent variable and dependent variable might actually relate in a logarithmic / exponential way (note: there are more formal ways to assess these kinds of models, we will cover those later in the class).

Weighted Least Squares Regression

Weighted least squares regression (WLS) is like the traditional regression that we have been doing, except in this case you can weight how much each observation contributes towards the estimation of the coefficients and the standard error(s). Thus far, each observation has contributed equally to generating the estimates. In WLS, you can define how much each observation "counts".

The weights used in WLS regression are specified by you.

Arbitrary Weights

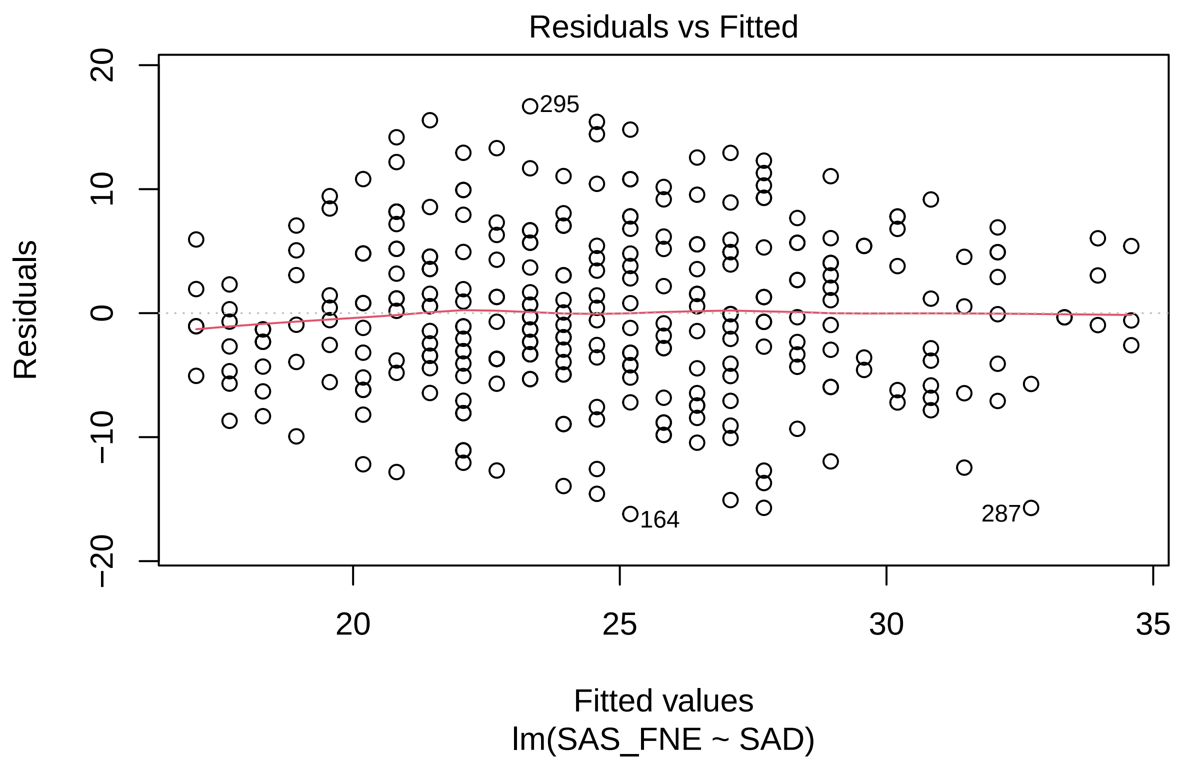
In some cases, you can just choose which weights to assign to the predictors. This is generally not recommended, because the weights are arbitrarily assigned. But for the sake of illustrating the intuition of WLS regression we will go over an example.

For this example, we will use the `phobia` data, and will predict `SAS_FNE` from `SAD`:

```
mod1 <- lm(SAS_FNE ~ SAD, data = phobia)
summary(mod1)
```

```
##
## Call:
## lm(formula = SAS_FNE ~ SAD, data = phobia)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.1975  -4.2823  -0.3366   4.9242  16.6808
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.05837    0.83194   20.50   <2e-16 ***
## SAD          0.62609    0.06058   10.33   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.673 on 300 degrees of freedom
## Multiple R-squared:  0.2625, Adjusted R-squared:  0.2601
## F-statistic: 106.8 on 1 and 300 DF,  p-value: < 2.2e-16
```

```
plot(mod1, which = 1)
```

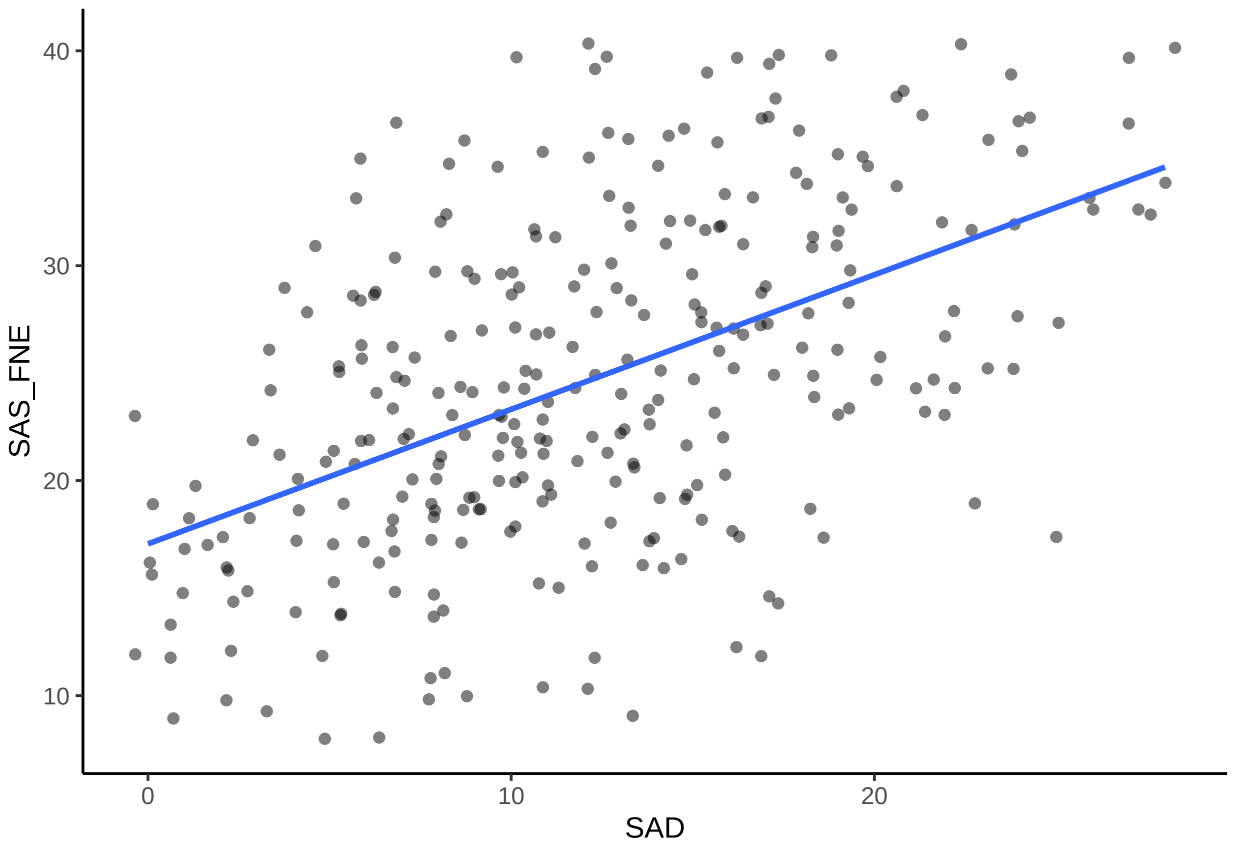


```
white(mod1)
```

```
## # A tibble: 1 x 5
##   statistic p.value parameter method       alternative
##       <dbl>   <dbl>     <dbl> <chr>        <chr>
## 1      7.55  0.0229         2 White's Test greater
```

```
ggplot(data = phobia,
       aes(x = SAD, y = SAS_FNE)) +
  geom_jitter(alpha = 0.5) +
  geom_smooth(method = "lm", se = F) +
  theme_classic()
```

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



Let's say that, for whatever reason, observations where `SAS_FNE` is greater than 30 shouldn't count as much in our regression. Let's say they should only contribute 50% as much information as all other observations. We could do a WLS regression and do exactly that.

```
# First, define a vector of weights
wts <- ifelse(phobia$SAS_FNE > 30, 0.5, 1)

cbind.data.frame(SAS_FNE = phobia$SAS_FNE,
                 wts) |>
  arrange(-SAS_FNE) |>
  head()
```

```
##   SAS_FNE wts
## 1      40 0.5
## 2      40 0.5
## 3      40 0.5
## 4      40 0.5
## 5      40 0.5
## 6      40 0.5
```

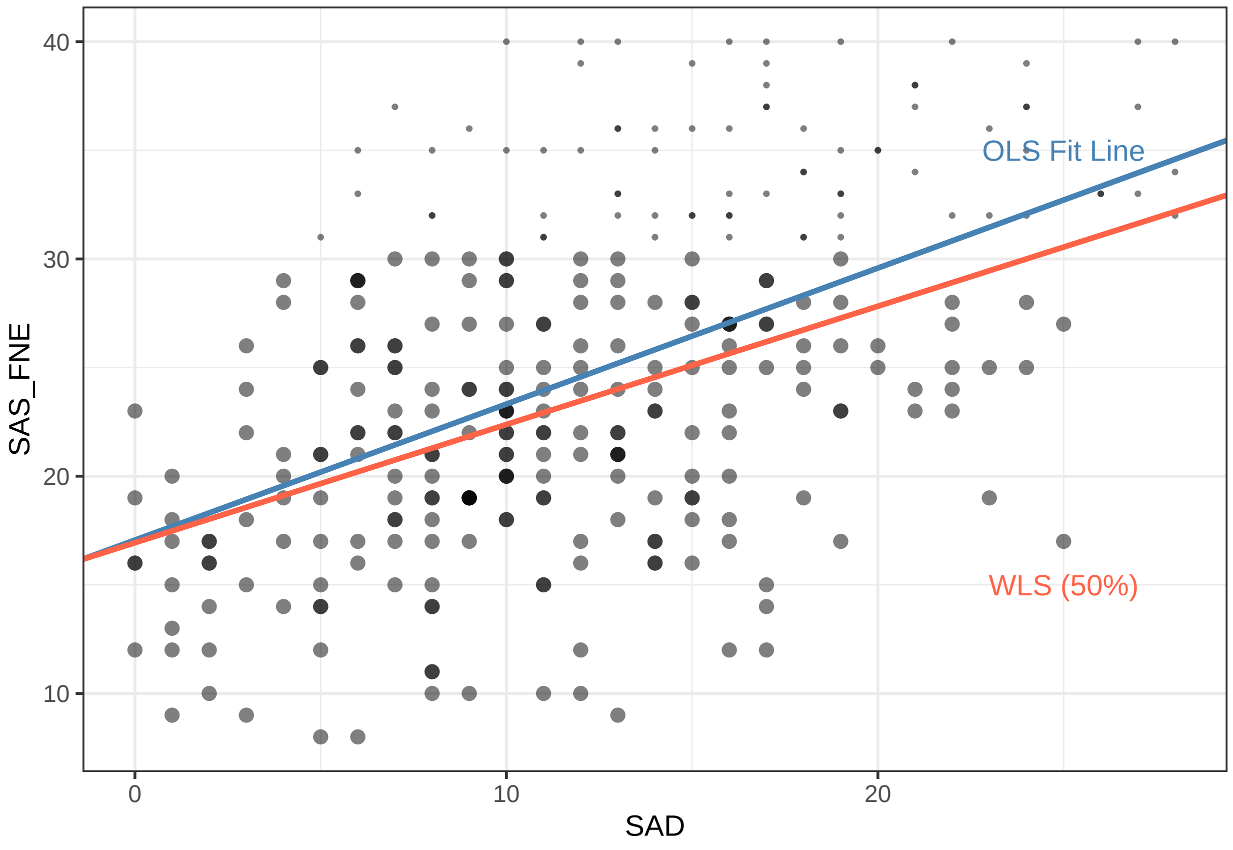
```
# Then, run the WLS regression
arbitrary_wls1 <- lm(formula = SAS_FNE ~ SAD,
                     data = phobia,
                     weights = wts)

summary(arbitrary_wls1)
```

```
##
## Call:
```

```
## lm(formula = SAS_FNE ~ SAD, data = phobia, weights = wts)
##
## Weighted Residuals:
##      Min       1Q   Median       3Q      Max
## -15.0130  -3.1774   0.8035   5.1218  12.4592
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 16.93676    0.76624  22.104   <2e-16 ***
## SAD          0.54432    0.05888   9.245   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.844 on 300 degrees of freedom
## Multiple R-squared:  0.2217, Adjusted R-squared:  0.2191
## F-statistic: 85.46 on 1 and 300 DF,  p-value: < 2.2e-16
```

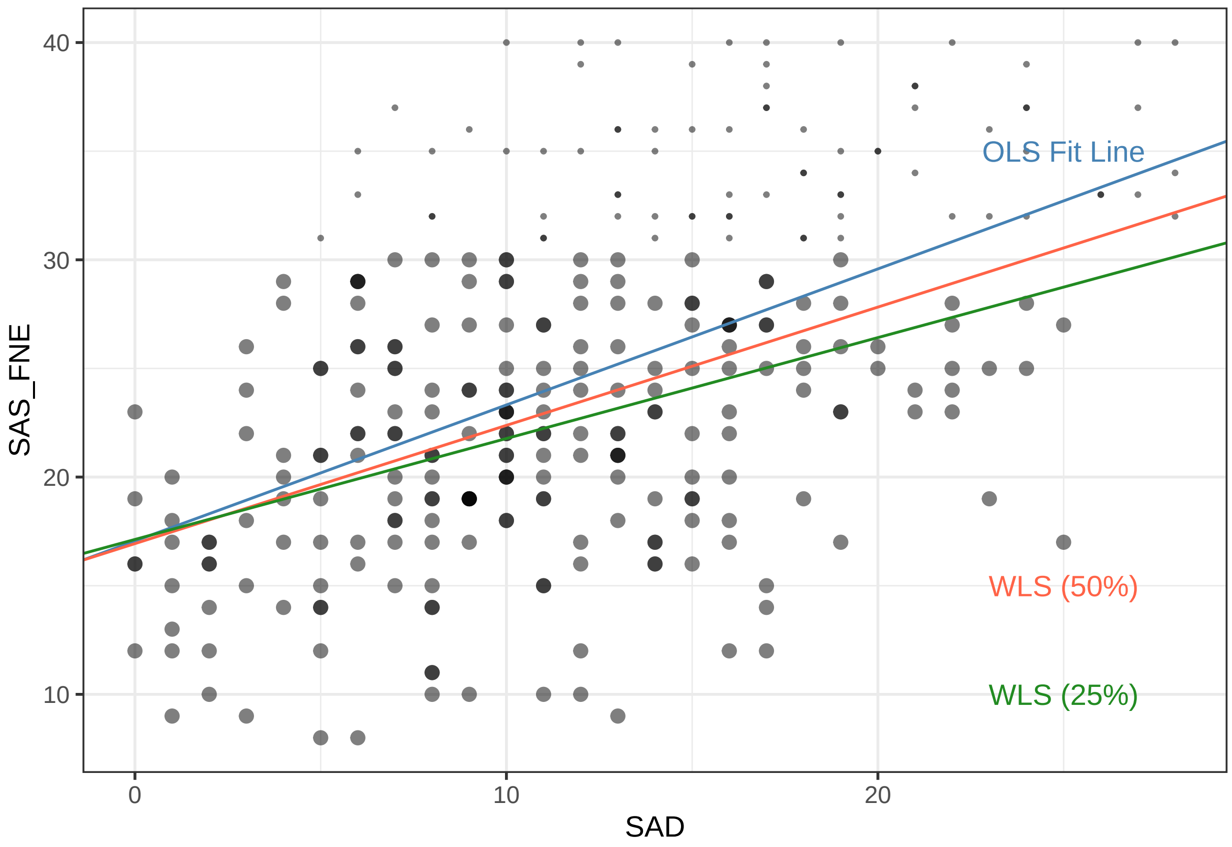
Let's compare the WLS regression we just did to the original OLS regression:

```
ggplot(data = phobia,
             aes(x = SAD, y = SAS_FNE)) +
      geom_point(alpha = .5, size = ifelse(phobia$SAS_FNE > 30, 0.5, 2)) +
      geom_abline(intercept = mod1$coefficients[1], slope = mod1$coefficients[2], color = "steelblue", 
      geom_abline(intercept = arbitrary_wls1$coefficients[1], slope = arbitrary_wls1$coefficients[2], c
      annotate(geom = "text", label = "OLS Fit Line", x = 25, y = 35, color = "steelblue", size = 4) +
      annotate(geom = "text", label = "WLS (50%)", x = 25, y = 15, color = "tomato", size = 4) +
      theme_bw()
```



What if we weighted the outliers at only 25%?

```r
wts2 <- ifelse(phobia$SAS_FNE > 30, 0.25, 1)
arbitrary_wls2<- lm(formula = SAS_FNE ~ SAD,data = phobia, weights = wts2)

ggplot(data = phobia,
             aes(x = SAD, y = SAS_FNE)) +
       geom_point(alpha = .5, size = ifelse(phobia$SAS_FNE > 30, 0.5, 2)) +
      geom_abline(intercept = mod1$coefficients[1], slope = mod1$coefficients[2], color = "steelblue") +
      geom_abline(intercept = arbitrary_wls1$coefficients[1], slope = arbitrary_wls1$coefficients[2], co
      geom_abline(intercept = arbitrary_wls2$coefficients[1], slope = arbitrary_wls2$coefficients[2], co
      annotate(geom = "text", label = "OLS Fit Line", x = 25, y = 35, color = "steelblue", size = 4) +
      annotate(geom = "text", label = "WLS (50%)", x = 25, y = 15, color = "tomato", size = 4) +
      annotate(geom = "text", label = "WLS (25%)", x = 25, y = 10, color = "forestgreen", size = 4) +
      theme_bw()
```



The graph shows that the less those some observations are weighted, the more the fit line moves away from them.

WLS with Inverse Variance Weights

The examples above were based on arbitrary weights — there was no reason why we chose those values to have the weights they did, and no reason why we chose the weights we did. You should not do this for analyzing your data. We just wanted to better understand WLS.

When we have heteroscedasticity (i.e., unequal variances in the residuals), OLS is no longer the most efficient estimator. Instead, WLS becomes the most efficient estimator, provided that the weights are equal to the inverse of the variance of the residuals

$$w_i = 1/\sigma^2_{\epsilon_i}.$$

Why do we use the inverse of the variance of the residuals? Because observations with high variance in the residuals are providing less information than observations with lower variance in the residuals, so by taking the inverse, observations with high variance / less information are weighted less than those with low variance / more information.

Let's walk through how we obtain these weights. We will continue with the same example.

1. Fit the OLS regression model without weights.
   - We already did this and called the regression object `mod1`.

```
summary(mod1)
```

```
##
## Call:
## lm(formula = SAS_FNE ~ SAD, data = phobia)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.1975  -4.2823  -0.3366   4.9242  16.6808
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.05837    0.83194   20.50   <2e-16 ***
## SAD          0.62609    0.06058   10.33   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.673 on 300 degrees of freedom
## Multiple R-squared:  0.2625, Adjusted R-squared:  0.2601
## F-statistic: 106.8 on 1 and 300 DF,  p-value: < 2.2e-16
```

2. Extract the absolute values of the residuals from our OLS model and regress them on the predicted values from the OLS model

```
# extract absolute residuals
abs_resids <- abs(residuals(mod1))
# extract predicted values
preds <- fitted(mod1)

wmod <- lm(abs_resids ~ preds)
```

3. From the secondary regression, extract the predicted values and square them. These squared values serve as estimates of the variance of the residuals, $\sigma^2_{\epsilon_i}$. We will take the inverse of these variances, and use them as weights in our weighted least squares regression.

```
# calculate variance weights
resid_vars <- fitted(wmod)^2
w <- 1/resid_vars

# WLS regression
wls_mod1 <- lm(SAS_FNE ~ SAD, data = phobia, weights = w)
wls_mod1
```

```
##
## Call:
```

```
## lm(formula = SAS_FNE ~ SAD, data = phobia, weights = w)
##
## Coefficients:
## (Intercept)          SAD
##     16.9949       0.6313
```

Why would we do this? The intuition is something like this. We believe that we have violated the homogeneity of variance assumption, and we would like to account for that. One way of doing so is by performing a WLS regression with weights equal to the inverse variance for each data point. So the above steps correspond to the following:

1. Fit your typical OLS regression
2. Extract the absolute residuals and predict these values using your original predictor. The (squared) predictions from this model serve as an estimate of the variance for each data point, $\sigma^2_{\epsilon_i}$.
3. Take the inverse variances and use these as weights in a WLS regression.

At the end of the day, much did our WLS change our results?

```
summary(mod1)
```

```
##
## Call:
## lm(formula = SAS_FNE ~ SAD, data = phobia)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.1975  -4.2823  -0.3366   4.9242  16.6808
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.05837    0.83194   20.50   <2e-16 ***
## SAD          0.62609    0.06058   10.33   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.673 on 300 degrees of freedom
## Multiple R-squared:  0.2625, Adjusted R-squared:  0.2601
## F-statistic: 106.8 on 1 and 300 DF,  p-value: < 2.2e-16
```

```
summary(wls_mod1)
```

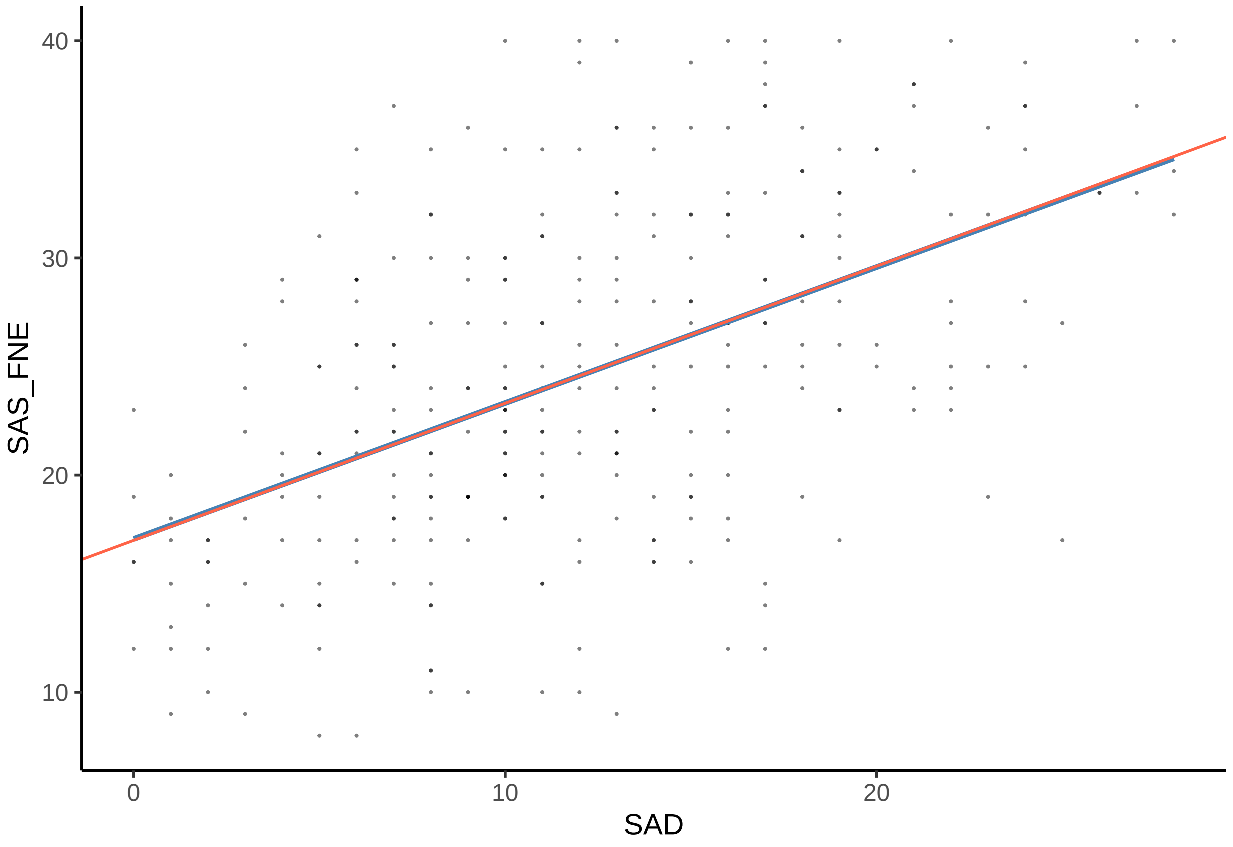
```
##
## Call:
## lm(formula = SAS_FNE ~ SAD, data = phobia, weights = w)
##
## Weighted Residuals:
##      Min       1Q   Median       3Q      Max
## -3.00087 -0.78259 -0.07047  0.93362  3.14774
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 16.99492    0.81108   20.95   <2e-16 ***
## SAD          0.63135    0.06107   10.34   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.241 on 300 degrees of freedom
## Multiple R-squared:  0.2627, Adjusted R-squared:  0.2602
## F-statistic: 106.9 on 1 and 300 DF,  p-value: < 2.2e-16
```

Our coefficient estimates hardly changed, but the residual standard error has been drastically reduced

We can also take a look at the fit lines

```
ggplot(data = phobia,
       aes(x = SAD, y = SAS_FNE)) +
  geom_point(alpha = .5, size = w) +
  theme_classic() +
  geom_smooth(method = "lm", se = F, fullrange = T, color = "steelblue") +
  geom_abline(intercept = wls_mod1$coefficients[1], slope = wls_mod1$coefficients[2], color = "tomato")
```

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



Note on Multiple Regression

Everything we talked about today is applicable to multiple regression. For example, if we fit a model predicted `SPAI_SP` from `FNE` and `SAD` (from the phobia data):

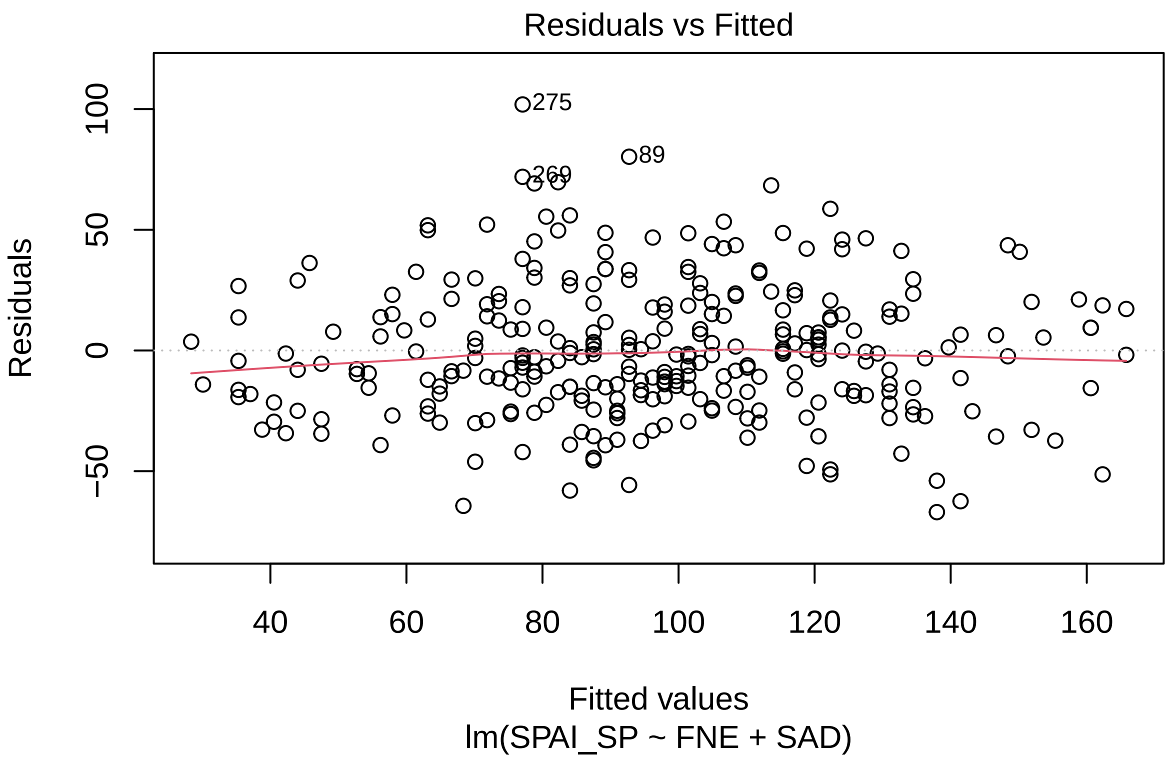
```
mod2 <- lm(SPAI_SP ~ FNE + SAD, data = phobia)
summary(mod2)
```

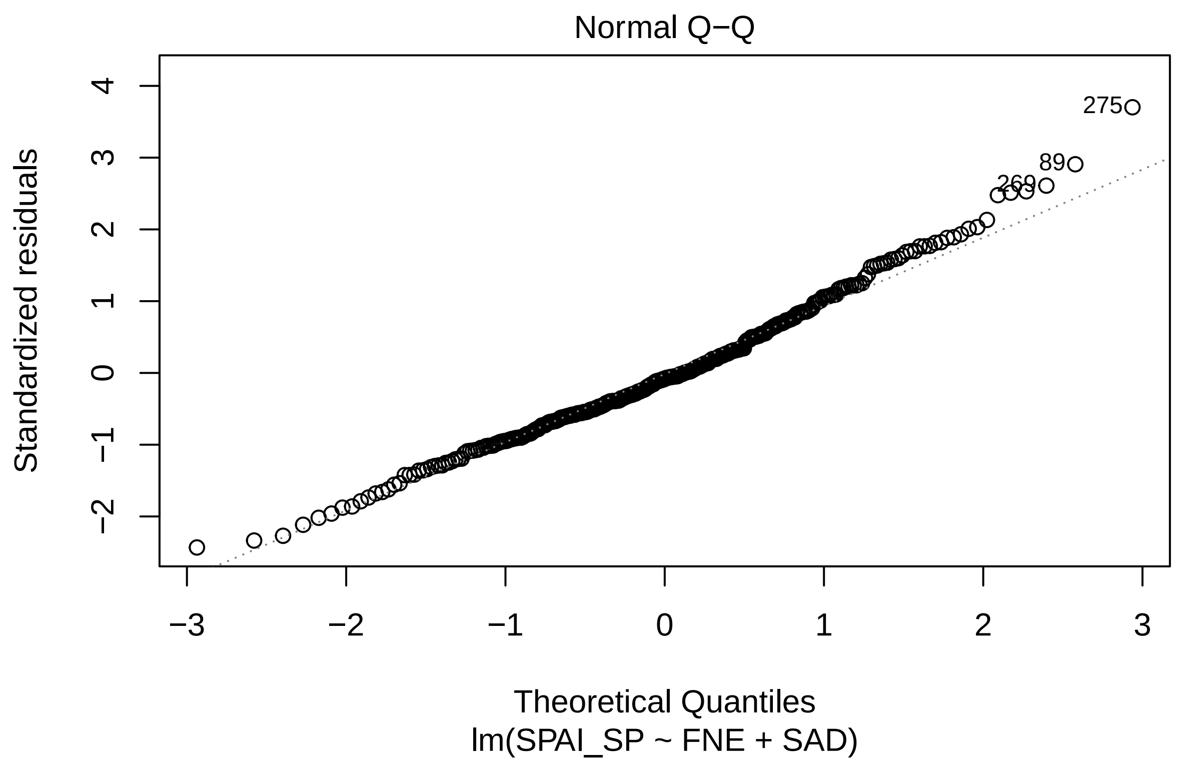
```
##
## Call:
## lm(formula = SPAI_SP ~ FNE + SAD, data = phobia)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max 
## -66.970 -18.015  -2.241  17.139 101.924 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  19.6594     5.2466   3.747 0.000215 ***
## FNE           1.7394     0.2862   6.077 3.72e-09 ***
## SAD           3.4803     0.2876  12.102  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 27.68 on 299 degrees of freedom
## Multiple R-squared:  0.5276, Adjusted R-squared:  0.5244 
## F-statistic:   167 on 2 and 299 DF,  p-value: < 2.2e-16
```

We could visually assess assumptions:

```
plot(mod2, which = c(1, 2))
```





We could perform the more formal asessments of assumptions

```
white(mod2)
```

```
## # A tibble: 1 x 5
##   statistic p.value parameter method       alternative
##       <dbl>   <dbl>     <dbl> <chr>        <chr>
## 1      3.31   0.508         4 White's Test greater
```

```
shapiro.test(mod2$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  mod2$residuals
## W = 0.98702, p-value = 0.00815
```

And we could perform a WLS multiple regression:

```
wls_reg <- lm(abs(residuals(mod2)) ~ fitted(mod2))
wts <- fitted(wls_reg)^2

wls_mod2 <- lm(SPAI_SP ~ FNE + SAD,
             data = phobia |> filter(!is.na(SPAI_SP) & !is.na(FNE) & !is.na(SAD)),
             weights = wts)

summary(wls_mod2)
```

```
##
## Call:
## lm(formula = SPAI_SP ~ FNE + SAD, data = filter(phobia, !is.na(SPAI_SP) &
##     !is.na(FNE) & !is.na(SAD)), weights = wts)
##
```

```
## Weighted Residuals:
##      Min       1Q   Median       3Q      Max
## -1457.22  -387.99   -48.28   375.67  2199.60
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  19.7211     5.2642   3.746 0.000215 ***
## FNE           1.7393     0.2867   6.067 3.93e-09 ***
## SAD           3.4755     0.2873  12.098  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 599.1 on 299 degrees of freedom
## Multiple R-squared:  0.5271, Adjusted R-squared:  0.524
## F-statistic: 166.7 on 2 and 299 DF,  p-value: < 2.2e-16
```

In practice, how much should you actually worry about these things?

Much and more has been written about the assumptions of linear regression, how to test them, how to fix you data to meet the assumptions when they are violated, etc. However, some methodologists have also said that you don't always need to worry about these assumptions. For example, according to section 3.6 in Data Analysis Using Regression and Multilevel/Hierarchical Models (Gelman and Hill, 2007), these are the assumptions you should worry about from most to least important

1. Validity
   - Do the data you are analyzing accurately map to the research question you are trying to answer? Seems trivial, but surprisingly difficult.
2. Additivity and linearity
   - The most important mathematical assumption of regression. Are your data additive (e.g., $y = a + b + c$), or multiplicative (e.g., $y = a \times b \times c$). Are you overlooking any interactions? Is the relationship between the predictors and the outcome linear (e.g., $y = x$) or do you need to model a nonlinear outcome (e.g., $y = x + x^2$)
3. Independence of errors
   - Are your observations independent? Usually something we simply assume by expertise with data, experimental design, etc., but in the case of temporal dependence (i.e., does ordering of your observations matter?), then we can test this using something like the ACF (see lecture notes.)
4. Equal variance of errors
   - When the variance of the residuals are unequal, WLS is more efficient. However, WLS does not affect the most important aspect of the regression model, the coefficient estimates $\hat{\beta}$. In most cases, heteroscedasticity this is a minor issue.
5. Normality of errors
   - Generally the least important of the assumptions. "In fact, for the purpose of estimating the regression line (as opposed to predicting individual points), the normality assumption is barely important at all. Thus, we do *not* recommend diagnostics of the normality of regression residuals".