Session 4

June 27, 2023

[1]:
```
df <- read.csv("../biostats bootcamp 2023.csv", stringsAsFactors = F)
df$race <- as.factor(df$race)
df$discharge_dispsn <- as.factor(df$discharge_dispsn)
```

1 Overview of Inference

Statistical inference mainly consists of two tasks: hypothesis testing and estimation.

1.1 Hypothesis Testing

We've spent some time covering hypothesis testing in the previous section. Hypothesis tests are centered around the idea of a null hypothesis and the choice of accepting or rejecting it based on the evidence your data provides.

The utility in this approach is to make *qualitative* claims about the world - an effect/association exists or it doesn't. Our only goal is providing an optimal guess between these two choices. However, one of the biggest drawbacks of hypothesis testing is that the null hypothesis is *never* exactly true. There usually exists some potentially extremely small effect/association, and if you increase your sample size enough, you will eventually detect that.

In basic and clinical science, this is sometimes not a major issue - it might be impossible with current infrastructure to collect enough data to get to that point. Also, the research question of interest is usually only concerned with this binary question of establishing an association. The mechanisms under study are complicated, and your research only serves to *structurally* establish a connection between two variables that you've isolated through your experimental design.

However, sometimes you find yourself in the scenario where your data is extremely large and a hypothesis test might be meaningless because it will always be significant. Sometimes, we already know or apriori believe associations exist, and our fundamental scientific question is centered around *quantifying* this relationship. This is where the statistical science of estimation comes into play.

1.2 Estimation

First - forget about all the null/alternative hypothesis stuff for a moment; we are no longer working in that framework. Keep your understanding of probability models in the back of your mind though.

We have already been doing the first part of estimation in Sessions 2 and 3. When you calculated all of those descriptive statistics, you were creating estimates. But now, we're in a position to look at this a little bit more abstractly. Let's take the example of when we calculated Spearman's correlation coefficient. If we had an infinite amount of data, we would have calculated a slightly

different coefficient than what we got, and we can consider this the *true* value. From our data, we tried to *estimate* the true value as closely as we could.

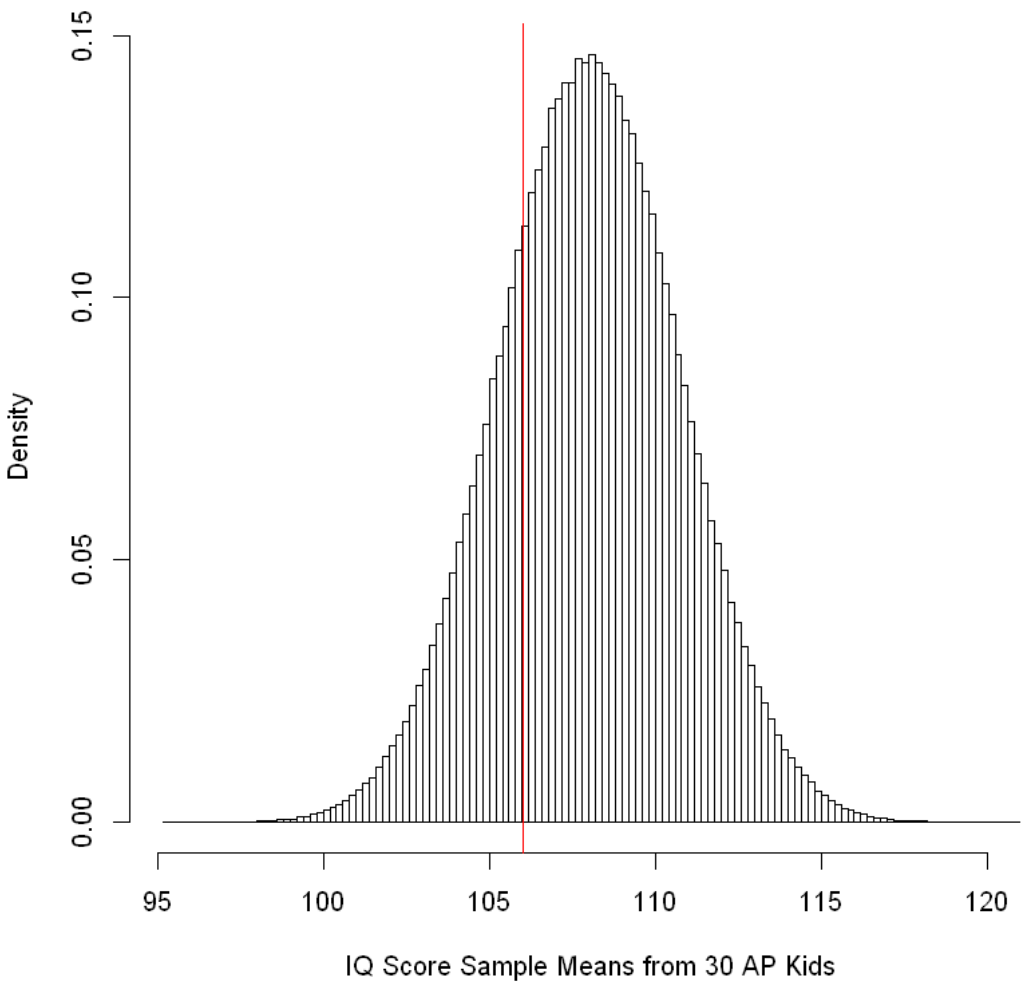
The sample median is an estimate of the true median. The sample Pearson Correlation Coefficient is an estimate of the true Pearson Correlation Coefficient. The sample difference of means is an estimate of the true difference of means. You get the idea. The true value would be scientifically useful to know, so we do our best with an estimate.

So how is this different from simple descriptive statistics? In inferential statistics, we are not satisfied with just forming an estimate. First, we are interested in the theoretical trade-offs between different algorithms to estimate things. In this course, lightly touching on the concept of *bias* will be all we have to say about this. Second, we must quantify the *uncertainty* around our estimate. This is where probability models come back into play and will be our focus.

1.2.1 Standard Error & Bias

To quantify uncertainty around an estimate, we have to play the infinite datasets experiment again. But now, the infinite datasets experiment cannot rely on some null probability model; it must rely on the true probability model from which our data was really generated! If we could do this, we could generate an estimate in each one of our infinite datasets and plot a histogram. Just to see this in action, let's look at the IQ example from the AP exam kids. Since I know this true distribution in this case (because I chose it), I can do this infinite datasets experiment.

[2]:
```
set.seed(12792) # ignore; here for reproducibility
x <- rnorm(30, mean = 108, sd = 15)
iq.sample.mean <- mean(x)
iq.est.distr <- rnorm(1000000, mean = 108, sd = 15/sqrt(30))
hist(iq.est.distr, probability = T, breaks = 100,
     xlab = 'IQ Score Sample Means from 30 AP Kids',
     main = '')
abline(v = iq.sample.mean, col = 'red')
```



This isn't the same type of plots we were showing last class.

- In the last class, this plot was the distribution of the test statistic when data is generated under a null distribution.

- In this class, this is the distribution of our estimate when data is generated under the true distribution. You can see this because the distribution is centered near 108, which is the true mean in this case. This distribution is referred to as the **sampling distribution**, but I will keep using the informal phrase "infinite datasets experiment".

I've drawn the location of our actual estimated sample mean in red to show that such a number is typical of this distribution. From this histogram that represents the infinite datasets experiment, we can now make an important inference.

We could figure out an **interval** that capture 95% of all possible sample estimates that we might

observe. For example, the interval starting at the 2.5th percentile and ending at the 97.5th percentile would accomplish this. For estimates that have an (approximately) normal distribution in the infinite datasets experiment, there is a shortcut formula for this.

$$[\mu - 1.96 * \text{SE}, \mu + 1.96 * \text{SE}]$$

where μ is the mean of this distribution and SE is its standard error. The standard deviation of the above histogram is known as the **standard error**. Standard deviation without qualification usually describes variation at the level of individuals; the name standard error is reserved for describing variation at the level of estimates/statistics. In this course, we will only be examining sampling distributions for the sample mean, and thus accordingly only discuss standard errors for the sample mean.

When do our estimates have (approximately) normal distributions in the infinite datasets experiment? Two scenarios: - When the underlying data on the individual level is approximately Normally distributed - When we have a lot of data, due to the Central Limit Theorem (CLT) kicking in

If our estimates have (approximately) normal distributions in the infinite datasets experiment, then the standard error can be calculated like

$$\text{SE} = \frac{\sigma}{\sqrt{n}}$$

where σ is the standard deviation of your individual-level data and n is your sample size.

Bias quantifies how far away the mean of this distribution is from the true value. I'm sure you've used bias colloquially all your life, but this is its formal quantitative definition. Ideally, we would like the bias to be 0. In the plot above, the bias is indeed 0. Do you recall the "correction factor" of $\frac{n}{n-1}$ in our sample variance estimator? This is there to ensure the estimate is unbiased; without this, the sample variance estimator would slightly underestimate the true variance on average.

This is one of those "theoretical properties" of estimation strategies, so we won't say much more on this topic.

1.2.2 Confidence Intervals

You might be thinking - that was a cute demonstration. But you were only able to do the infinite datasets experiment, since you knew the true distribution to start with! You'll never be able to do that in real life! And right you are. So how do we get around this?

Similarly to how we navigated things in hypothesis testing, we make near assumptions, and we make close approximations. For example, suppose that my observed sample mean and sample standard devation in the IQ example were called $\hat{\mu}$ and $\hat{\sigma}$. From the get-go - we're assuming that these AP kids have an IQ distribution that is

$$\text{Normal}(\mu, \sigma)$$

for *some* true mean μ and *some* true standard deviation σ. To get the standard error we just discussed, we need to know the true σ, but what if we just assumed $\hat{\sigma}$ was close enough to σ and continued from there? In other words, we pretend the data was generated by

$$\text{Normal}(\mu, \hat{\sigma})$$

If we are trying to make an inference for the true mean, we can use the formula from the last section to get the standard error for the mean of this distribution. If you have a lot of data, this approach of using $\hat{\sigma}$ approximately works even if the original data was not normally distributed, since the CLT kicks in.

If we proceed this way, we have made potentially 2 approximatation errors. First, we calculated the standard error for Normal$(\mu, \hat{\sigma})$ instead of Normal(μ, σ). Second, we assumed the distribution of our estimate was Normal, when it is usually only approximately normal. (However, it is exactly Normal in this one example by construction). These errors justify placing a little hat on our standard error to acknowledge this, giving us the formula

$$\hat{\text{SE}} = \frac{\hat{\sigma}}{\sqrt{n}}$$

Remember that interval we constructed earlier that captures 95% of estimates we would observe in the infinite datasets experiment? Skipping the math, we can actually flip this on its head and create an interval dependent on our estimate $\hat{\mu}$ that will likely capture the true μ at a 95% rate. This is known as a 95% confidence interval. Using the formula from before (which, again, assumes Normality of the infinite-datasets distribution), when we flip this on its head we get

$$\text{CI}_{95\%} = [\hat{\mu} - 1.96 * \text{SE}, \hat{\mu} + 1.96 * \text{SE}]$$

As the true standard error is unavailable to us, we can approximate this confidence interval with

$$\text{CI}_{95\%} = [\hat{\mu} - 1.96 * \hat{\text{SE}}, \hat{\mu} + 1.96 * \hat{\text{SE}}]$$

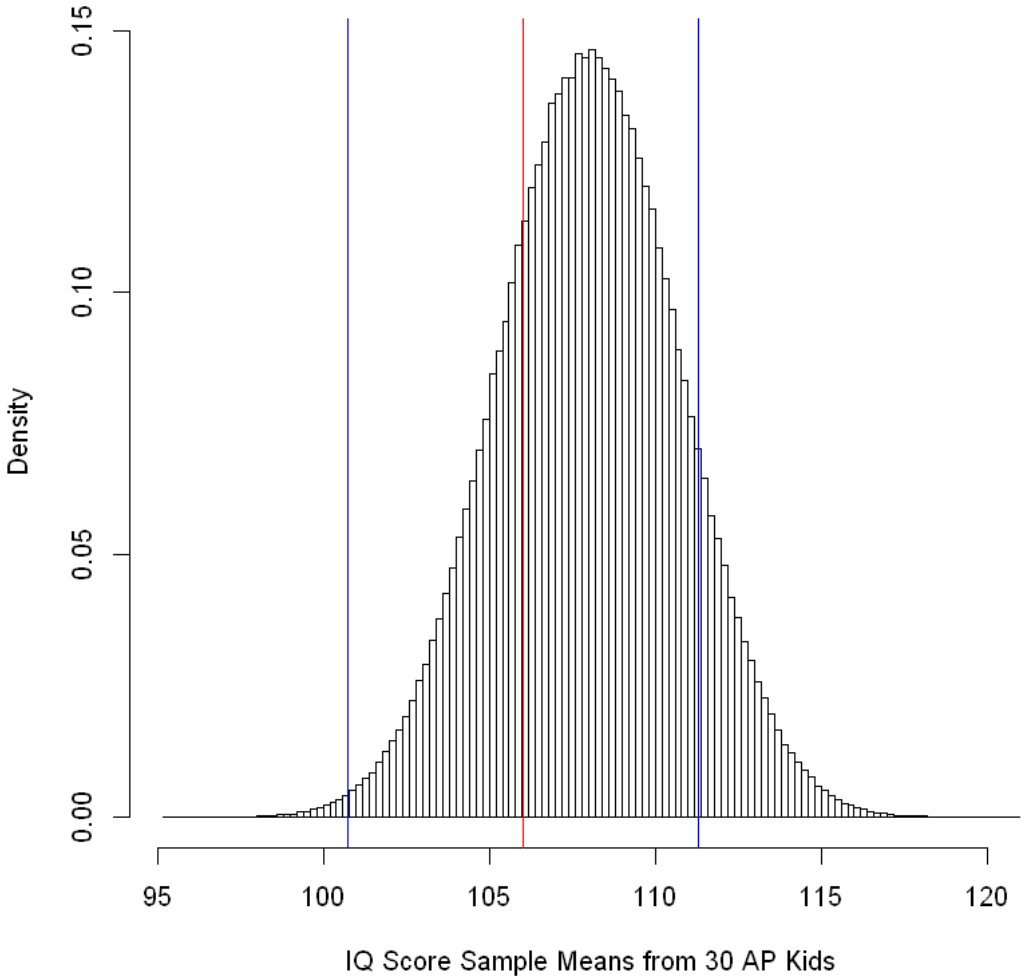
which is known as a 95% "Wald" Confidence Interval.

How can you interpret your confidence interval? Weirdly enough, it's kind of a "meta" claim about the algorithm to construct the confidence interval. It says that 95% of the time that you construct this type of confidence interval, it will capture the true value you were originally trying to estimate. "Capture" means that that the true value falls within the interval. This is slightly different than the incorrect claim that we might opt to make - that there is a 95% chance the true value is in the interval. The true value is not random! The interval is, since it's based on our data. Thus, you can never make a probabilistic claim about your particular interval; you can only make a claim about how these type of intervals perform in the long-run over many studies.

Let's construct a CI for the IQ example. We'll print and plot it. The confidence interval bounds will be in blue and our original sample mean will be in red.

[3]:
```
iq.sample.sd <- sd(x)
std.error.hat.of.mean <- iq.sample.sd/sqrt(30)
lower.bound <- iq.sample.mean - 1.96*std.error.hat.of.mean
upper.bound <- iq.sample.mean + 1.96*std.error.hat.of.mean
print(c(lower.bound, upper.bound)) # display interval numerically
hist(iq.est.distr, probability = T, breaks = 100,
     xlab = 'IQ Score Sample Means from 30 AP Kids',
     main = '')
abline(v = iq.sample.mean, col = 'red')
abline(v = lower.bound, col = 'blue')
abline(v = upper.bound, col = 'blue')
```

```
[1] 100.7621 111.2869
```



Recall that 108 was the true mean, so this particular interval did happen to correctly capture the true mean, but we would never actually know this in a real study. What we would know is that is that if we kept repeating the experiment, the red line (and thus the blue lines) would jump around from experiment to experiment; in 95% of these experiments, the space between the blue lines would capture the true mean.

In-Class Exercise 1: Gaining Confidence

1. Generate a "dataset" consisting of 200 samples from a Gamma distribution with the following code:
   x <- rgamma(200, shape = 6.5, rate = 0.1)

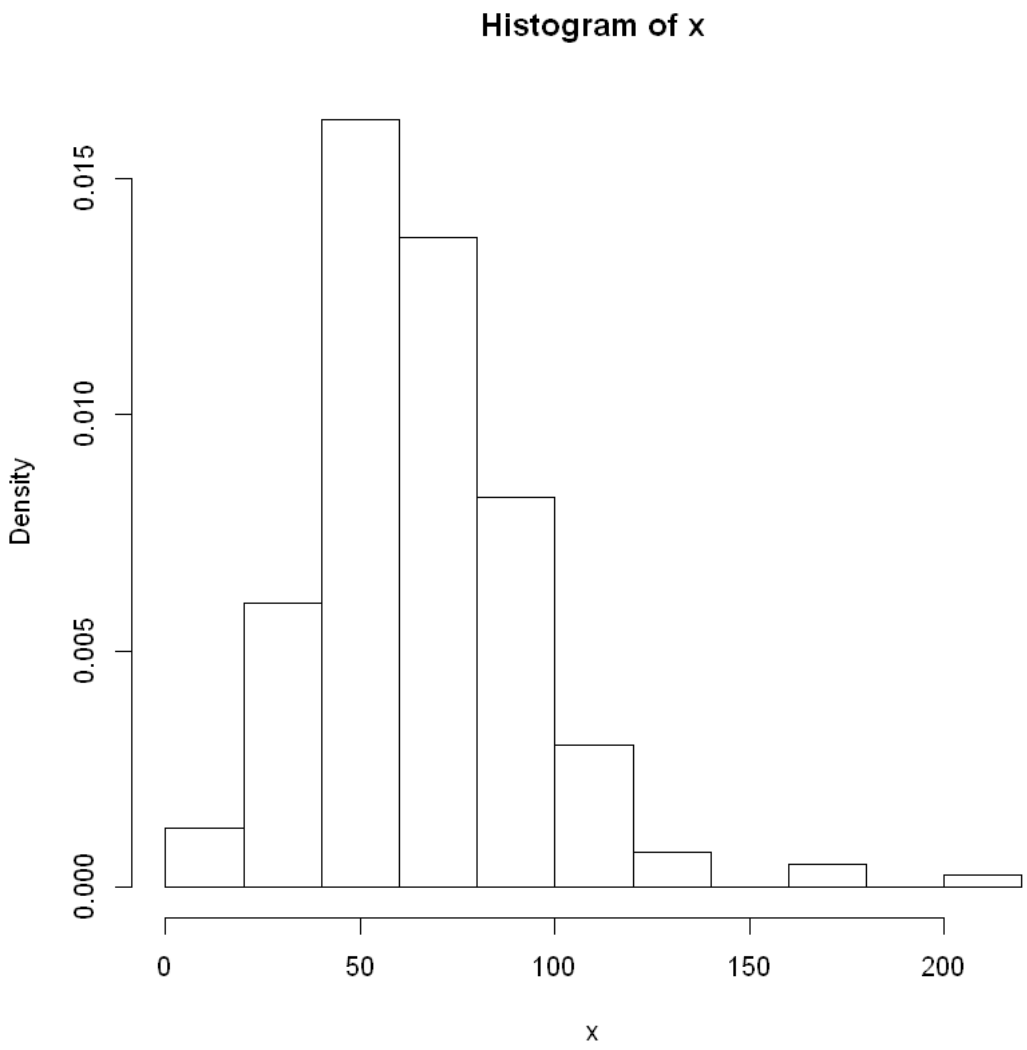
Plot a histogram to visualize this data and then construct a 95% Wald confidence interval for the true mean. Simply derive the lower and upper bounds of the interval in separate calculations.

The true mean of the distribution we sampled from is 65; did your interval capture it?

2. Define "x" as the difference in systolic_max and diastolic_max blood pressure in our dataset. Produce a 95% Wald confidence interval for the true mean of the distribution that governs x.

[4]:
```r
# generate data and plot
set.seed(31185)
x <- rgamma(200, shape = 6.5, rate = 0.1)
hist(x, prob = T)

# calculate CI
x.sample.mean <- mean(x)
n <- length(x)
std.err.hat <- sd(x)/sqrt(n)
lower.bound <- x.sample.mean - 1.96*std.err.hat
upper.bound <- x.sample.mean + 1.96*std.err.hat

# print
print(paste("Sample Mean:", round(x.sample.mean, 3)))
CI.string <- paste0("[", round(lower.bound, 3), ", ", round(upper.bound, 3),␣
 ↪"]")
print(paste("Confidence Interval:", CI.string))
```

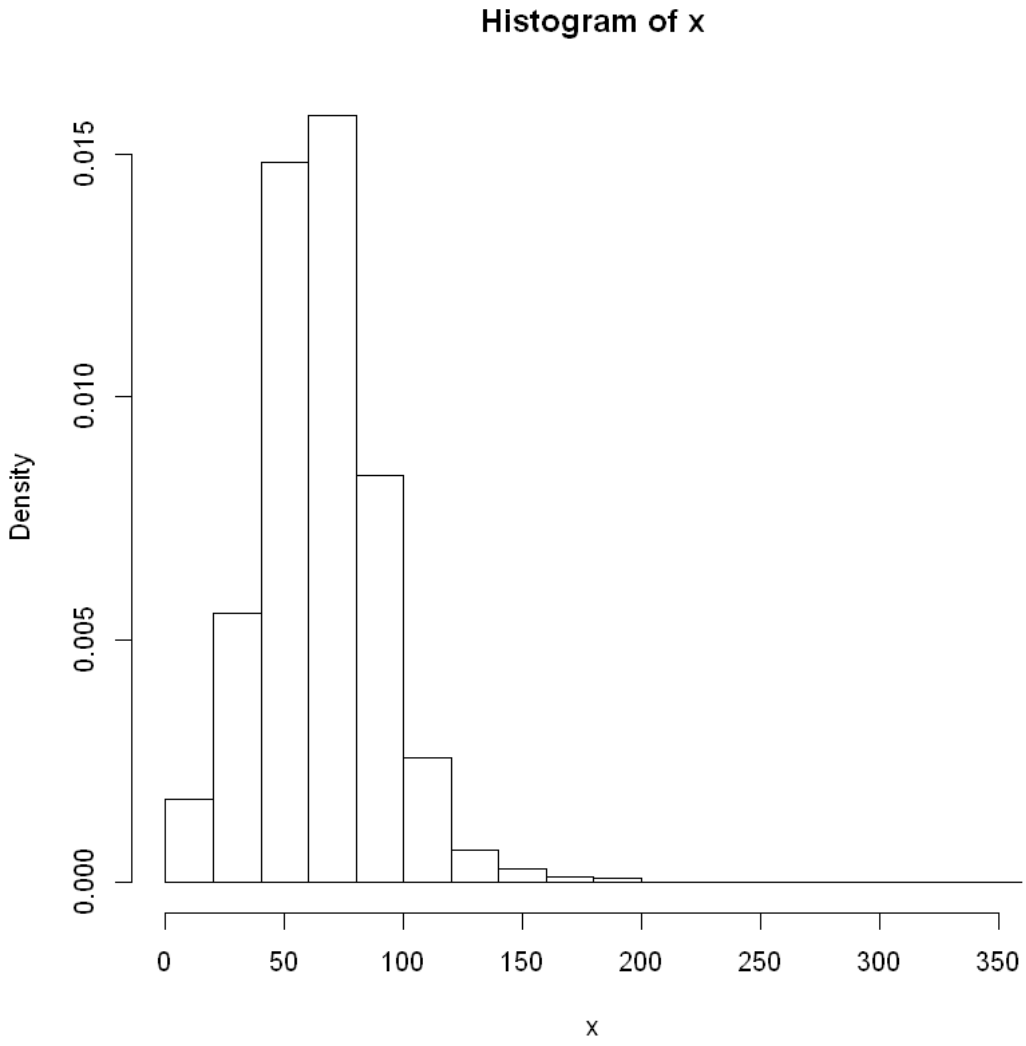
```
[1] "Sample Mean: 66.111"
[1] "Confidence Interval: [62.248, 69.973]"
```



[5]:
```r
x <- df$systolic_max - df$diastolic_max
hist(x, prob = T)

# calculate CI; same exact code as last time
x.sample.mean <- mean(x)
n <- length(x)
std.err.hat <- sd(x)/sqrt(n)
lower.bound <- x.sample.mean - 1.96*std.err.hat
upper.bound <- x.sample.mean + 1.96*std.err.hat

# print
print(paste("Sample Mean:", round(x.sample.mean, 3)))
```

```
CI.string <- paste0("[", round(lower.bound, 3), ", ", round(upper.bound, 3),␣
  ↪"]")
print(paste("Confidence Interval:", CI.string))
```

```
[1] "Sample Mean: 65.108"
[1] "Confidence Interval: [64.24, 65.976]"
```



1.2.3 Prediction

Let's say you were analyzing a problem where a dependent variable is being associated with one (or more) independent variables.

After you have performed estimation, you can use then re-use your estimates to perform **prediction** for new patients. How do you define an optimal prediction? There are actually several ways but

we'll focus on the most common definition for the duration of this session. In our definition, an **optimal prediction** is the **mean** outcome of the subgroup of individuals who look exactly like you with respect to the independent variables.

Let's look at a quick example. Suppose creatinine_max is the dependent variable and consult_nephrology is the independent variable. I can calculate the optimal predictions for the maximum creatinine of new patients who got a consult and new patients who didn't as follows:

[6]:
```
round(mean(df$creatinine_max[df$consult_nephrology == 0]), 3)
round(mean(df$creatinine_max[df$consult_nephrology == 1]), 3)
```

1.767

5.165

I can equivalently express this idea in an equation as follows:

$$\hat{y} = \hat{\beta}_0(1 - x) + \hat{\beta}_1 x$$

In this case, - $\hat{y}$ represents the prediction for creatinine_max - x represents consult_nephrology coded as 0/1 - $\hat{\beta}_0 = 1.76658$ - $\hat{\beta}_1 = 5.16466$

We have little hats on our β's, since the optimal prediction would be based on the true means, which we are approximating with the sample means from our data.

2 The Basic Linear Model

2.1 Structure & Assumptions

A linear model is a specific type of probability model. In its most typical formulation, it treats some independent variables (aka predictors) as fixed and *non-random*. These variables can be of any type: binary, categorical, ordinal, or continuous.

The model then treats a dependent variable or outcome y as *random*. This variable must be continuous (but we'll extend this to model binary and count outcomes in the next session). In this model, the independent variables help predict the mean of y for a particular setting of x, i.e. the optimal prediction, using a linear equation. Again, let's use an example: - y will be log(wbcc) - x will be race

How do you make a mathematical equation out of a categorical variable? I can't do Black$*3$. There are many approaches to fix this, but the simplest is called *one-hot* encoding. This approach creates several new binary variables, one for each category, capturing whether that category is active for that patient. Here is a display of what this looks like. Interpret the columns as x_1, x_2, x_3, x_4 and interpret the rows as different patients.

[7]:
```
contr.treatment(levels(df$race), contrasts = F)
```

	Asian	Black	Other	White
Asian	1	0	0	0
Black	0	1	0	0
Other	0	0	1	0
White	0	0	0	1

From this, we can make a linear equation:

$$\mu_y(x) = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4$$

where $\mu_y(x)$ is the true mean of y for any particular setting of x. This is a theoretical model, since we use unknown true quantities of β. Inferring what these true quantities might be and quantifying our uncertainty around them will be the goal of our analysis. When we replace these with our sample estimates, we get a more practical prediction equation like we looked at before

$$\hat{y} = \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \hat{\beta}_3 x_3 + \hat{\beta}_4 x_4$$

But for clarity in what's happening, let's focus on the theoretical model for a moment.

The first assumption the linear model makes is that the *remaining* variation after you make your optimal prediction is Normally distributed and centered at 0. Let's review what this means. Suppose I made a log(wbcc) prediction for Asian people. That prediction is not going to be spot-on for everyone. There's going to be a certain amount of error or variation around my prediction in that subgroup. The *distribution* of that error is what we're describing here and referring to as remaining variation. It's the variation in the outcome that remains after we've accounted for being Asian. This remaining variation should have a mean of 0. This should be intuitive - if your prediction is optimal (under our mean definition), the average error should be 0, right?

The second assumption that the linear model makes is that the standard deviation is the *same* across all levels of x.

….wait. Aren't these the exact same assumptions we made for our Student's t / ANOVA tests? Indeed. These are special cases of the more general linear model!

Returning to the assumptions, if we combine the insight from the second assumption with the first, we can infer that all errors in a linear model, regardless of the level of x, collectively follow a single Normal distribution. This gives rise to a different way of expressing a linear model that is quite common to see:

$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \epsilon$$

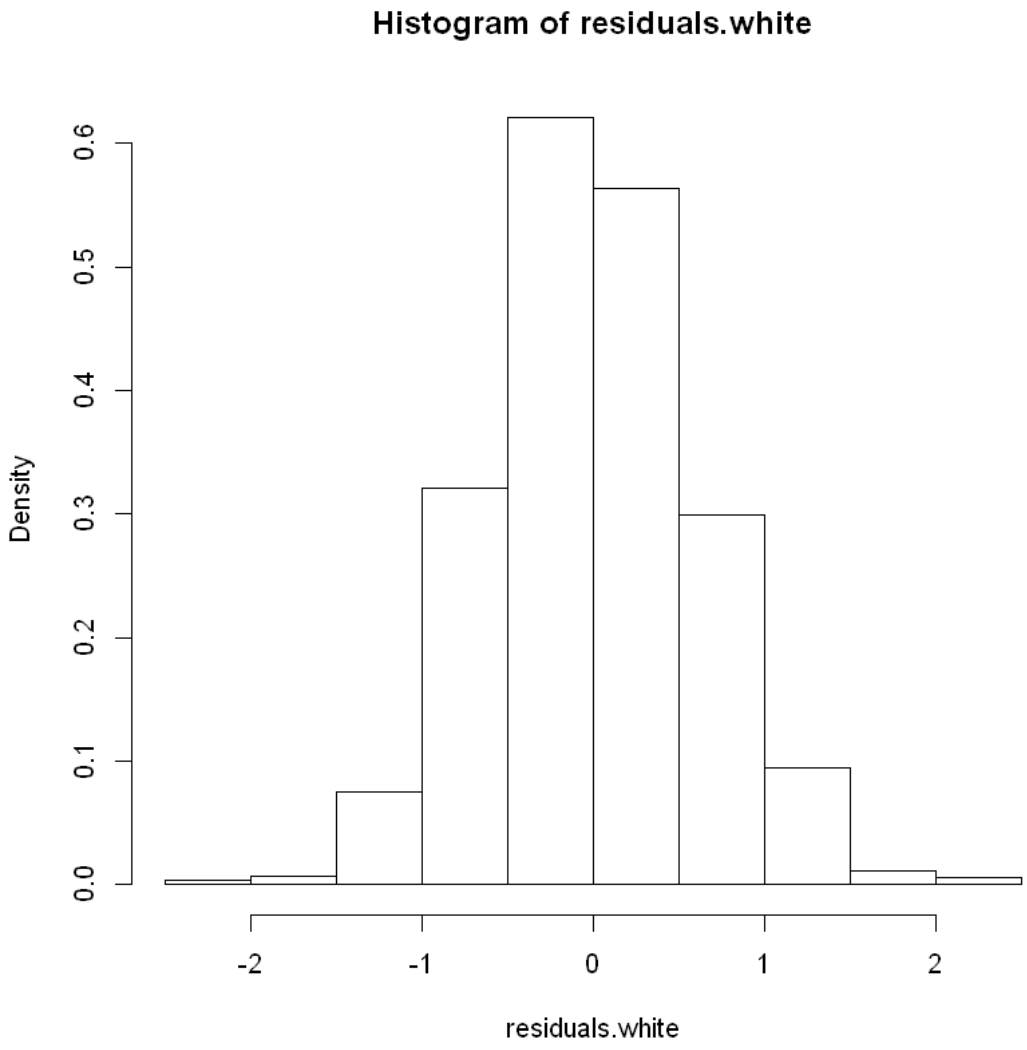
where ϵ has a Normal distribution centered at 0 and with some unknown (but common to all x) standard deviation. In this interpretation, y's are generated by first making the optimal linear prediction using x and then performing a random Normally distributed fluctuation away from this value.

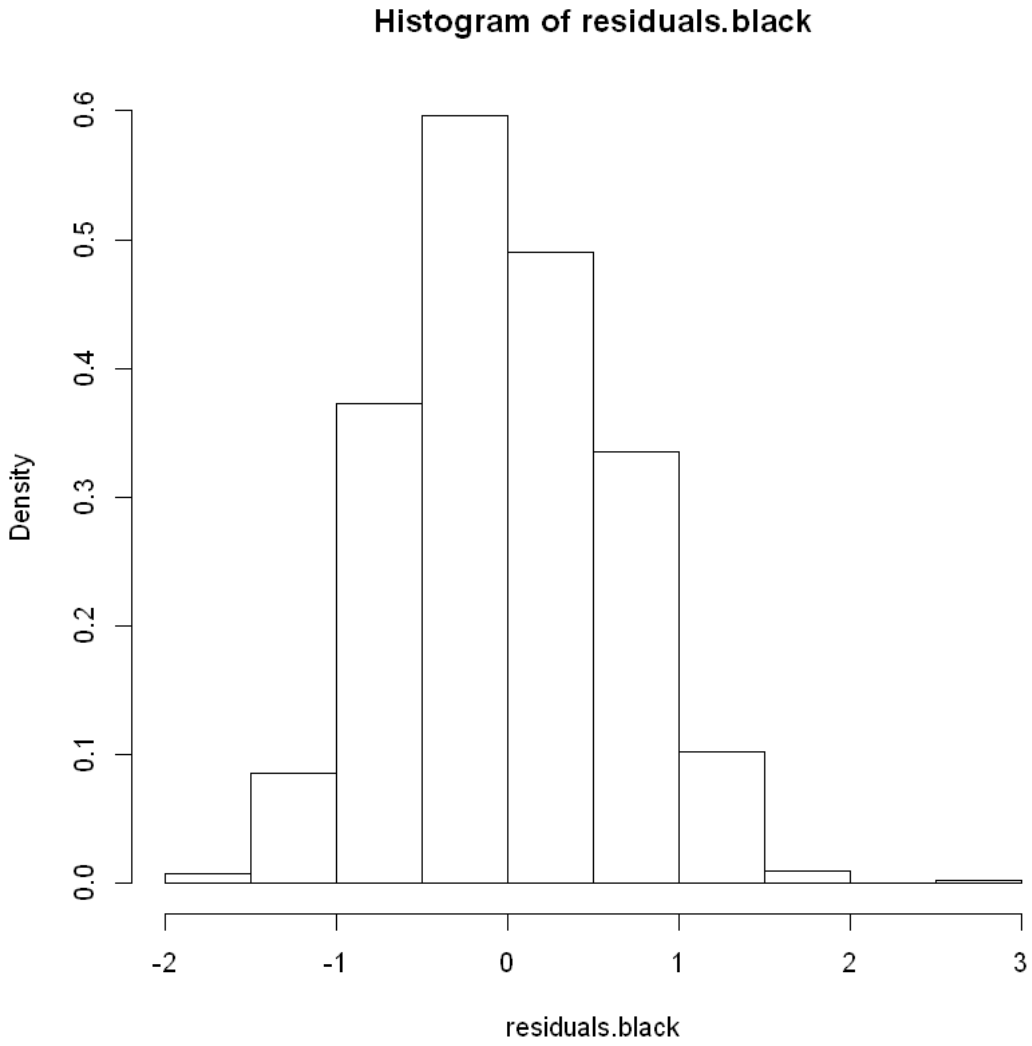
There are actually a few more technical assumptions that need to be satisfied, but we'll ignore them for this session.

We already ensured in the last session that log(wbcc_max) was Normally distributed within each race, but we'll display the residuals for two races in a histogram to reinforce the perspective we're building.

[8]:
```
# subset outcome to each group
y.white <- log(df$wbcc_max)[df$race == 'White']
y.black <- log(df$wbcc_max)[df$race == 'Black']
y.hat.white <- mean(y.white)
y.black.white <- mean(y.black)

# subtract off predictions to get remaining variation
```

```r
residuals.white <- y.white - y.hat.white
residuals.black <- y.black - y.black.white

# visualize
hist(residuals.white, prob = T)
hist(residuals.black, prob = T)
```





These seem approximately Normal with similar standard deviations.

Since the assumptions are approximately satisfied, the linear model is appropriate for modeling this relationship. There are two things you should keep in mind: - y does not have to be *overall* Normally distributed. Only the *remaining* variation after accounting for x should be Normally distributed. The classic example of this is the variation of human height after accounting for sex. - Even if the assumptions are satisfied, this does not mean the linear model is *the* one and only true model that describes how your data was generated. The linear model just provides *a* correct probabilistic description for how your data could have been generated. As long as we have *a* correct probabilistic description, we can make accurate inferences.

2.2 Running the Analysis

Okay, let's go over the syntax to actually create the linear model and infer some estimates for our β's. This will return the results in a list. The list has several names that we could use to extract particular parts of the result using our indexing patterns. The "-1" part of our formula tells R that I don't want to include an intercept in my model. At least not for now - more on this soon.

[9]:
```
model <- lm(log(wbcc_max) ~ -1 + race, data = df)
print(names(model))
```

```
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "contrasts"     "xlevels"       "call"          "terms"
[13] "model"
```

2.2.1 Reading the Output

However, the most useful parts of the result can be attained by calling the "summary" function on the list that was output. Compare the coefficient estimates to simply computing the means in each subgroup to see that they match:

[10]:
```
summary(model) # linear model
aggregate(log(wbcc_max) ~ race, data = df, FUN = mean) # subgroup means
```

```
Call:
lm(formula = log(wbcc_max) ~ -1 + race, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4296 -0.4264 -0.0273  0.4326  4.5025

Coefficients:
          Estimate Std. Error t value Pr(>|t|)
raceAsian  2.66523    0.07619   34.98   <2e-16 ***
raceBlack  2.53690    0.02105  120.54   <2e-16 ***
raceOther  2.63466    0.02858   92.18   <2e-16 ***
raceWhite  2.61188    0.01379  189.46   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.619 on 3412 degrees of freedom
Multiple R-squared:  0.9463,        Adjusted R-squared:  0.9463
F-statistic: 1.504e+04 on 4 and 3412 DF,  p-value: < 2.2e-16
```

race	log(wbcc_max)
Asian	2.665228
Black	2.536896
Other	2.634655
White	2.611880

There are several key results displayed here.

- On top, a 5-number summary is given of the prediction errors also known as **residuals**. These should look approximately Normally distributed as a consequence of the errors at each level of x having the same Normal distribution. You can use this 5-number summary as a quick and dirty check, or you can be more formal about this and use "qqnorm(model$residuals)". Note that passing this check does not *ensure* the assumptions are satisfied, but are rather a mininimum prerequisite to them potentially being satisfied.
- In the middle section, we are given *estimates* for the true values of our β coefficients, which are the primary targets of our analysis. To the right, we are given estimates for the standard errors of each $\hat{\beta}$. Recall that these each have their own distributions under the infinite datasets experiment. To the right of that, we are given a t-test statistic and a p-value for the null hypothesis that the corresponding true β coefficient is 0 (alternative is not 0).
- At the bottom, we are given a few extra pieces of less important information. Residual standard error gives us an estimate for what that common unknown value of standard deviation is in our Normally distributed prediction errors. (Adjusted) R-squared gives a value between 0 and 1 that estimates how predictive your model is. It describes the percentage of the variation in y that was accounted for by x; as you can see, our model was not very predictive at all. Lastly, the F-statistic and corresponding p-value give the results for a hypothesis test for the null hypothesis that *all* the β coefficients except the Intercept's are collectively 0. Recall that, here, we do not have an intercept.

2.2.2 Generating Confidence Intervals

The estimated coefficients that we saw were listed along side with *standard errors*. If your sample size is large enough, you can use this information to provide a 95% "Wald" confidence interval. For example, a 95% Wald confidence interval for β_1 would look like

$$\text{CI}(\beta_1) = \left[\hat{\beta}_1 - 1.96 * \hat{\text{SE}}_1, \hat{\beta}_1 + 1.96 * \hat{\text{SE}}_1\right]$$

We could pull the standard errors out of the model output and manually construct this. However, there is a more convenient way to do this!

[11]:
```
confint(model)
```

	2.5 %	97.5 %
raceAsian	2.515844	2.814613
raceBlack	2.495632	2.578160
raceOther	2.578616	2.690694
raceWhite	2.584851	2.638909

2.2.3 Creating Predictions

Once we have a model object, we can use the "predict" function on it to predict the mean of the outcome on old or new data. The "predict" function will recognize the model as a linear model and will redirect it to the "predict.lm" function. So if you want to use the help tool, look at ?predict.lm. Let's predict on some new data. We can also request confidence intervals and prediction intervals from this function. We'll look at confidence intervals first.

[12]:
```
new.data <- data.frame(race = c("Black", "Black", "Other", "Asian", "Other"))
predict(model, new.data, interval = "confidence")
```

fit	lwr	upr
2.536896	2.495632	2.578160
2.536896	2.495632	2.578160
2.634655	2.578616	2.690694
2.665228	2.515844	2.814613
2.634655	2.578616	2.690694

As you recall, 95% confidence interval is a procedure to construct an interval that hopes to capture some "true" value. If we performed this procedure in millions of datasets, we would expect the capture to be successful 95% of the time. Let's look at the first row, corresponding to somebody who is Black. This confidence interval is trying to capture the "true" mean log(wbcc_max) value for Black people under this particular linear probability model. In the third row, this confidence interval is trying to capture the "true" mean log(wbcc_max) value for someone who is of race "Other" in this particular linear probability model, and so on.

Now, we'll repeat the prediction, but this time ask for prediction intervals.

[13]:
```
predict(model, new.data, interval = "prediction")
```

fit	lwr	upr
2.536896	1.322588	3.751204
2.536896	1.322588	3.751204
2.634655	1.419756	3.849555
2.665228	1.442463	3.887994
2.634655	1.419756	3.849555

You might notice these intervals are way wider. These can be interpretted as windows where you are likely to see future data. So for example, the first row tells us that for all black people, we would expect to see 95% of all their data in the interval provided.

2.3 A Binary/Categorical Predictor

We ran our first linear model including this curious "-1" term in the formula, which tells R to not use an intercept when fitting the model. What would happen if we didn't include that and decided to include an intercept?

2.3.1 Dummy Coding

Note that if we don't use the summary function, R will just print the estimated coefficients.

[14]:
```
lm(formula = log(wbcc_max) ~ race, data = df)
```

```
Call:
lm(formula = log(wbcc_max) ~ race, data = df)

Coefficients:
(Intercept)    raceBlack    raceOther    raceWhite
    2.66523     -0.12833     -0.03057     -0.05335
```

As we can see, R took the "Asian" coefficient out and made it the baseline (i.e. associated with the intercept). Every other race's coefficient now describes how different their mean is *compared* to the Asian mean. For example, the black mean is $2.66523 - 0.12833$.

This is the default behavior in R and corresponds to a coding scheme for the categorical variable known as "dummy coding" aka "treatment contrasts". The idea is that there is always some reference or "control" which becomes associated with the intercept and all other coefficients represent differences from that baseline. For example, if you have several drugs in your model, each that is either administered or not, it makes a lot of sense to have a common intercept coefficient associated with taking *no* drug, and separate coefficients associated with the administration of each individual drug.

Due to this perspective usually being necessary with respect to at least one variable in your model, you will almost always have an intercept. For mathematical reasons, it is not possible to include an intercept *and* every level of a categorical variable as a predictor. So the most common approach is to treat one level as a reference. Just to see, this is how race is coded under treatment contrasts:

[15]:
```
contr.treatment(levels(df$race))
```

	Black	Other	White
Asian	0	0	0
Black	1	0	0
Other	0	1	0
White	0	0	1

As just described, there is no binary marker for "Asian" since it is the reference, and an Asian patient would have a 0 for each of the other binary markers. So in this case, the categorical race variable only gets transformed into 3 binary variables, x_1, x_2, x_3, corresponding to Black, Other, and White.

However, suppose that we wanted to choose the reference level ourselves. Perhaps to the majority population in this demographic? How could we do that?

[16]:
```
df$race <- relevel(df$race, ref = "White")
contr.treatment(levels(df$race))
```

	Asian	Black	Other
White	0	0	0
Asian	1	0	0
Black	0	1	0
Other	0	0	1

[17]:
```
lm(formula = log(wbcc_max) ~ race, data = df)
```

```
Call:
lm(formula = log(wbcc_max) ~ race, data = df)

Coefficients:
(Intercept)    raceAsian    raceBlack    raceOther
    2.61188      0.05335     -0.07498      0.02277
```

Now "Asian" has its own coefficient and "White" is associated with the intercept.

2.3.2 In-Class Exercise 2: Baby's First Regression

Run a linear model where log(wbcc_max) is the dependent variable and discharge_dispsn is the independent variable and save the results to a variable called "model".

1. Produce a summary for your model using the summary function. What level of the discharge variable is associated with the intercept? Using the levels function on the discharge column might help here.
2. What is the optimal predicted mean log(wbcc_max) for those discharged to a Skilled Nursing Facility? Calculate this directly from the estimated model coefficients.
3. Produce confidence intervals for your estimates using the confint function. State an interpretation for the intercept's confidence interval.

Solution:

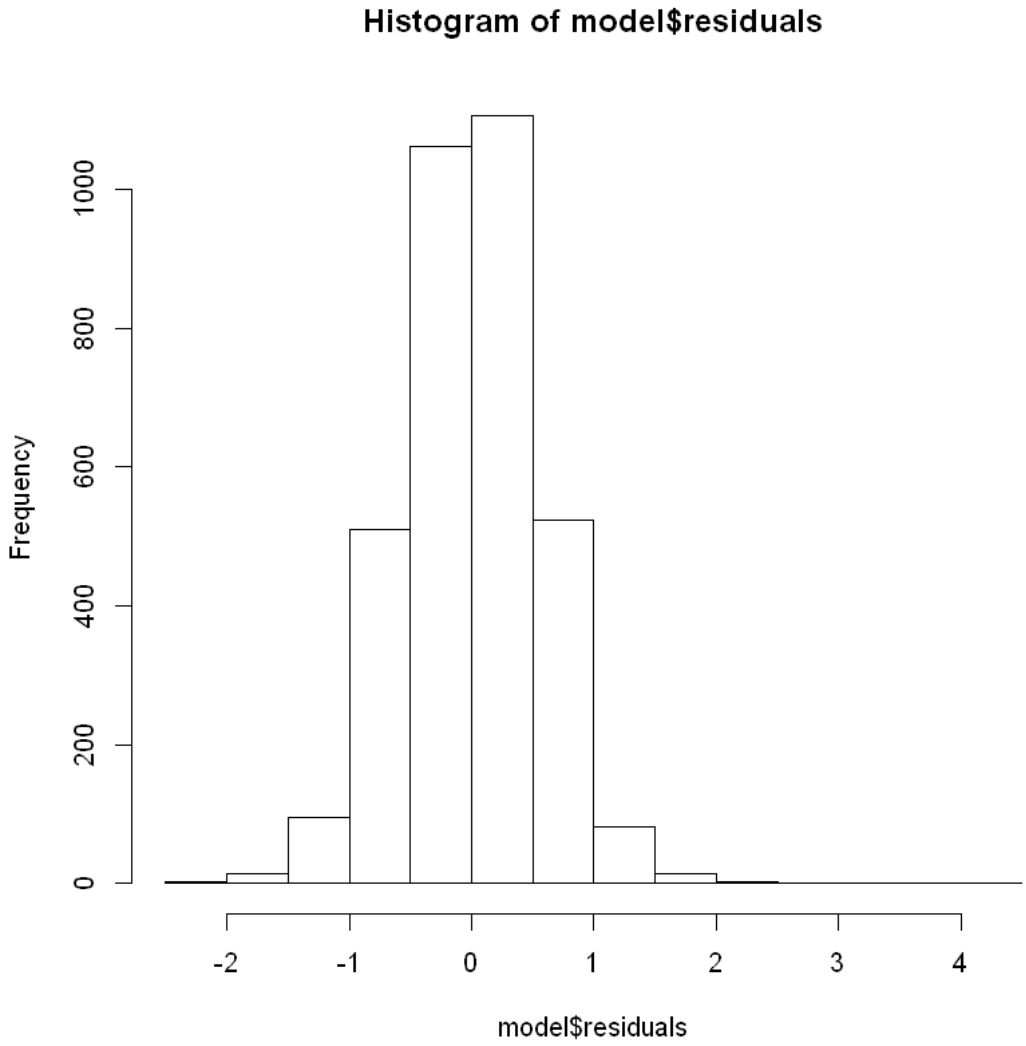
[18]:
```
model <- lm(formula = log(wbcc_max) ~ discharge_dispsn, data = df)

hist(model$residuals) # visualize residuals

summary(model) # print summary
levels(df$discharge_dispsn) # first level is associated with intercept
# can also tell, since that is the only level not given in the model␣
 ↪coefficient names

confint(model) # confidence intervals
```

```
Call:
lm(formula = log(wbcc_max) ~ discharge_dispsn, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
```

```
-2.1715 -0.3659  0.0070  0.3745  4.1171

Coefficients:
                                    Estimate Std. Error t value Pr(>|t|)
(Intercept)                          2.35382    0.01415 166.301  < 2e-16
discharge_dispsnHospice              0.42716    0.05535   7.717 1.55e-14
discharge_dispsnOther                0.66617    0.02388  27.899  < 2e-16
discharge_dispsnSkilled Nursing Facility  0.24539    0.02335  10.509  < 2e-16

(Intercept)                         ***
discharge_dispsnHospice             ***
discharge_dispsnOther               ***
discharge_dispsnSkilled Nursing Facility ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5587 on 3412 degrees of freedom
Multiple R-squared:  0.1882,        Adjusted R-squared:  0.1875
F-statistic: 263.6 on 3 and 3412 DF,  p-value: < 2.2e-16
```

1. 'Home' 2. 'Hospice' 3. 'Other' 4. 'Skilled Nursing Facility'

	2.5 %	97.5 %
(Intercept)	2.3260657	2.3815680
discharge_dispsnHospice	0.3186359	0.5356884
discharge_dispsnOther	0.6193567	0.7129890
discharge_dispsnSkilled Nursing Facility	0.1996057	0.2911683



1. Discharge to home is associated with the intercept.
2. 2.35382 + 0.24539 (i.e. model\$coefficients[1] + model coefficients[4])
3. The interval [2.326, 2.382] provides us with a range in which we believe the true value of the intercept lies. This belief is made quantitative by the claim that 95% of such intervals correctly capture their true value.

2.3.3 Deviation Coding

What if it really just does not make sense to set a reference level? What if instead of comparing each level to a reference level, you are more interested in comparing each level to some overall mean across the levels? Try coding your variables using "deviation coding" aka "sum contrasts".

[19]:
```
contr.sum(levels(df$race))
```

White	1	0	0
Asian	0	1	0
Black	0	0	1
Other	-1	-1	-1

I'll explain the negatives shortly. Sum contrasts do two important things. First, they associate the intercept with the mean of the category-specific means; a common misconception is that they associate the intercept with the overall mean. Secondly, they don't have a specific coefficient for the *last* level, in this case "Other". Let's run it and interpret.

[20]:
```
result <- lm(formula = log(wbcc_max) ~ race, data = df,
             contrasts = list(race = "contr.sum"))
result
```

```
Call:
lm(formula = log(wbcc_max) ~ race, data = df, contrasts = list(race = "contr.
  ↪sum"))

Coefficients:
(Intercept)        race1        race2        race3
  2.6121650   -0.0002847    0.0530634   -0.0752688
```

Let's examine this output. First, we'll verify the intercept is indeed the mean of means.

[21]:
```
race.means <- aggregate(log(wbcc_max) ~ race, data = df, FUN = mean, na.action␣
  ↪= na.omit)[["log(wbcc_max)"]]
names(race.means) <- levels(df$race)
mean.of.means <- mean(race.means)
round(mean.of.means, 6)
```

2.612165

Next, we'll show that each coefficient is the difference between one of the race's means and the mean-of-means, excluding race's last level.

[22]:
```
round(race.means[1:3] - mean.of.means, 6)
```

White -0.000285 **Asian** 0.053063 **Black** -0.075269

Lastly, we can see the mean of "Other" is actually hidden inside of the coefficients we got. This is what those negative 1's above were encoding: the intercept minus all the other coefficients produces the mean for "Other".

[23]:
```
race.means[4]
result$coefficients[[1]] - sum(result$coefficients[2:4])
```

Other: 2.63465511866033

2.63465511866033

Okay how is this useful? Well particularly if you're trying to run a hypothesis test for a level's coefficient, and you want to examine whether that level's value is signficantly different from the mean-of-means.

[24]:
```
levels(df$race)
summary(lm(formula = log(wbcc_max) ~ race, data = df,
           contrasts = list(race = "contr.sum")))
```

1. 'White' 2. 'Asian' 3. 'Black' 4. 'Other'

```
Call:
lm(formula = log(wbcc_max) ~ race, data = df, contrasts = list(race = "contr.
  ↪sum"))

Residuals:
    Min      1Q  Median      3Q     Max
-2.4296 -0.4264 -0.0273  0.4326  4.5025

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.6121650  0.0212940 122.671  < 2e-16 ***
race1       -0.0002847  0.0234192  -0.012  0.99030
race2        0.0530634  0.0579307   0.916  0.35974
race3       -0.0752688  0.0259788  -2.897  0.00379 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.619 on 3412 degrees of freedom
Multiple R-squared:  0.003463,        Adjusted R-squared:  0.002587
F-statistic: 3.953 on 3 and 3412 DF,  p-value: 0.007953
```

As we can see from the results, the mean for "Black" (race3) does seem statistically sigificantly different from the mean-of-means. We arrived at a similar conclusion using an ANOVA test last class. There are many more types of coding schemes you can use in R, each affording a different interpretation of the coefficients. In fact, there exists a coding scheme which gives us back the exact same results as ANOVA (though the p-values presented are a bit different due to different null/alternative hypotheses). However, a linear model allows us to do way more then ANOVA.

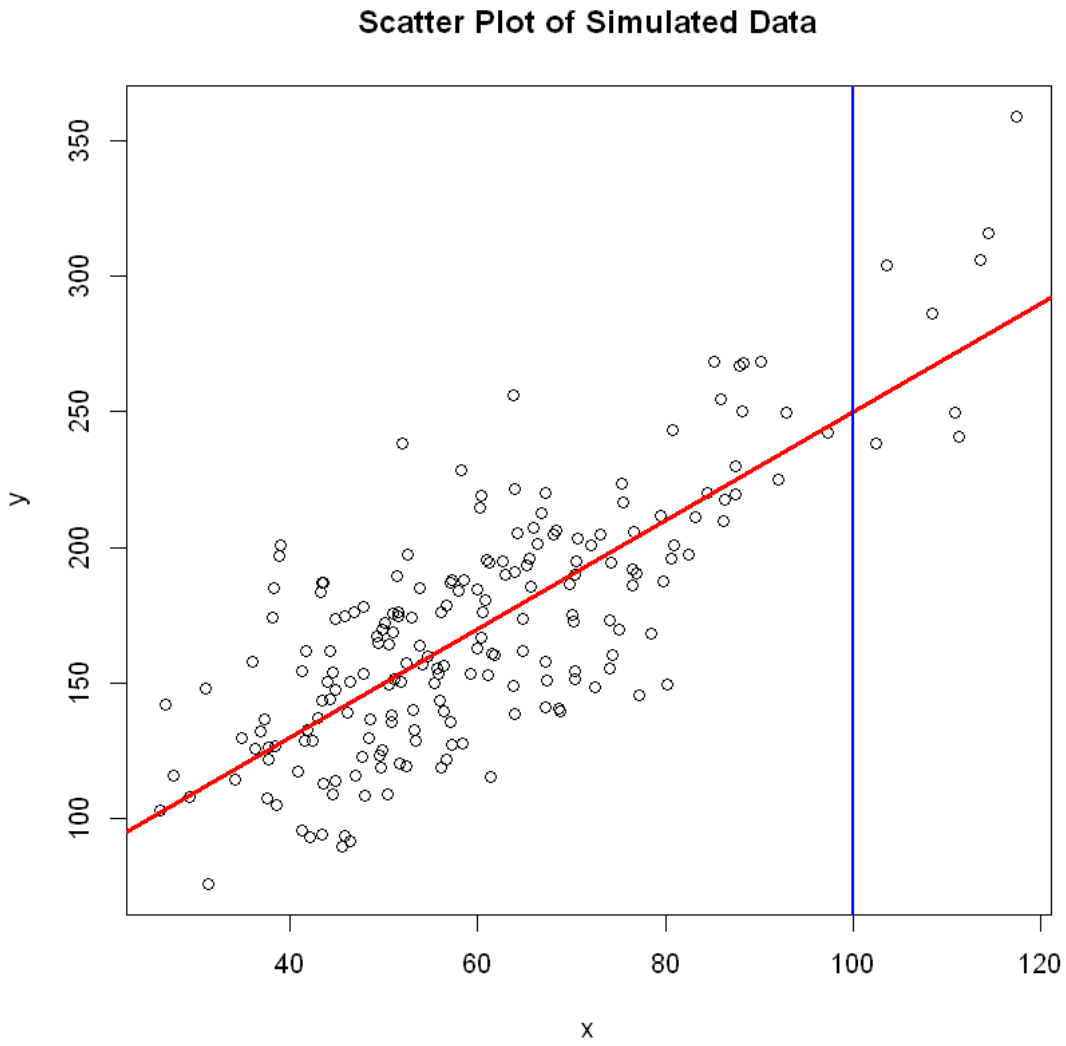
2.4 A Continuous Predictor

We will next examine a linear model with a single continuous predictor, bicarbonate_max, which we will call x. Thus in this case the mean equation looks like:

$$\mu_y(x) = \beta_0 + \beta_1 x$$

Here's a quick visualization of what data that satisfies a linear model should look like with a single continuous predictor.

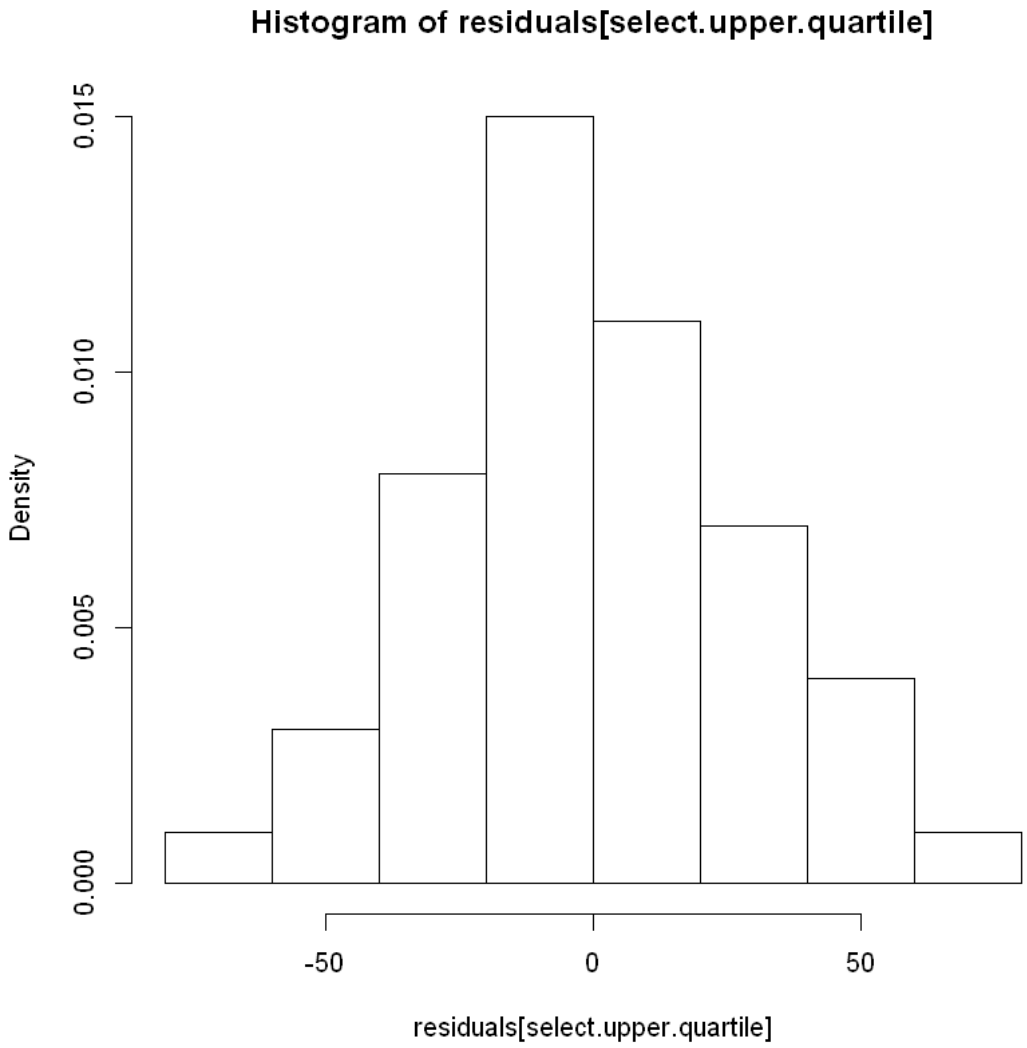
[25]:
```r
set.seed(2113)
n <- 200
x <- rgamma(n, 12, 0.2)
mu.y <- 50 + 2*x
y <- mu.y + rnorm(n, 0, 30)
plot(x, y, main = "Scatter Plot of Simulated Data")
abline(a = 50, b = 2, col = 'red', lwd = 3)
abline(v = 100, col = 'blue', lwd = 2)
```



As you can see, it will be hard to confirm the model assumptions in this case. Let's take the level $x = 100$. At this level, there are no datapoints! How can we confirm the residuals are Normally distributed at $x = 100$? One approach might be to collectively look at all levels near $x = 100$. This could be achieved by binning, for example. We could sort the data according to x and then bin it

into, say, 4 groups of 50. Within each group, we could check the residuals. Let's look at this for the last group.

[26]:
```
residuals <- y - mu.y
select.upper.quartile <- x > quantile(x, 0.75)
hist(residuals[select.upper.quartile], prob = T)
```



A little skewed, but if I were doing some sort of model checking, I'd say this was good enough.

Okay, let's get into a real example! We will try to predict the log of the maximum white blood cell count, so this will be our y. We'll use bicarbonate_max to inform our predictions.

[27]:
```
lm(formula = log(wbcc_max) ~ bicarbonate_max, data = df)
```

```
Call:
lm(formula = log(wbcc_max) ~ bicarbonate_max, data = df)

Coefficients:
    (Intercept)  bicarbonate_max
          1.738            0.029
```

These results are telling us that we are estimating the true β_0 with $\hat{\beta}_0 = 1.738$, and we are estimating the true β_1 with $\hat{\beta}_1 = 0.029$. However, the intercept coefficient is not the best for interpretation. As it stands right now, the intercept describes the mean when someone has a bicarbonate_max of 0. Then the coefficient for bicarbonate_max describes how much the mean changes for every additional unit of bicarbonate_max from 0.

2.4.1 Centering and Scaling Predictors

We can change the interpretation of the intercept coefficient by first *centering* our continuous predictor, i.e. subtracting the sample mean. We can change the interpretation of our bicarbonate_max coefficient by *scaling* our continuous predictor, i.e. dividing by the sample standard deviation.

To do this, we're going to use the **scale** function, which can actually do both! It takes in 2 logical arguments after the variable: whether to center or not and whether to scale or not. We'll first center but not scale.

[28]:
```
model <- lm(log(wbcc_max) ~ scale(bicarbonate_max, T, F), data = df)
model
```

```
Call:
lm(formula = log(wbcc_max) ~ scale(bicarbonate_max, T, F), data = df)

Coefficients:
                  (Intercept)  scale(bicarbonate_max, T, F)
                        2.597                         0.029
```

We can see that the estimated coefficient for bicarbonate max is unchanged; accordingly, its interpretation has not changed either. However, the intercept now represents the mean of log(wbcc_max) when someone has a bicarbonate_max equal to its sample average. This is much more meaningful. The intercept represents a baseline predicted log wbcc mean for the average person (in terms of bicarbonate_max values) and the estimated coefficient for bicarbonate_max tells us how to modify this prediction based on each unit deviated from the average.

We could also fit our model after both centering and scaling. This changes the coefficient for bicarbonate_max to represent how much our prediction for the mean log(wbcc_max) would change with each standard deviation from the sample mean bicarbonate_max.

[29]:
```
lm(formula = log(wbcc_max) ~ scale(bicarbonate_max, T, T), data = df)
```

```
Call:
```

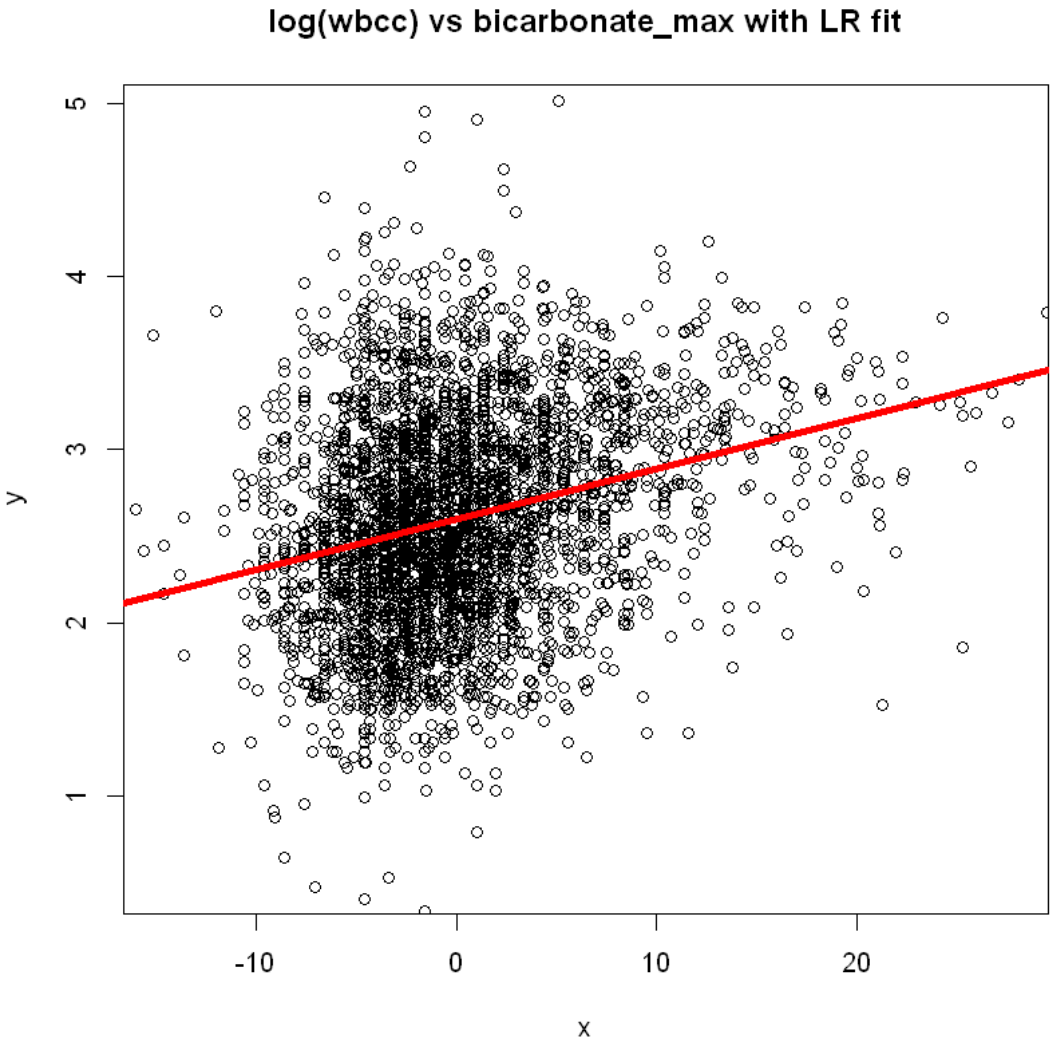
```
lm(formula = log(wbcc_max) ~ scale(bicarbonate_max, T, T), data = df)

Coefficients:
                  (Intercept)  scale(bicarbonate_max, T, T)
                       2.5971                        0.1674
```

As you can see, the intercept is unchanged but now the coefficient for bicarbonate_max now has a much larger value, since a standard deviation of bicarbonate_max is larger than 1. However, we're going to go back to the model where we only centered, since there are many cases where this setup affords the more useful interpretation.

2.4.2 Visualization & Model Diagnostics

Here is a plot of the line we estimated in that model; note that bicarbonate_max was first centered. We can extract the intercept and slope using the "coefficients" name in the model list. We can plot the line using the **abline** function.

[30]:
```
x <- scale(df$bicarbonate_max, T, F)
y <- log(df$wbcc_max)
plot(x, y,
     xlim = quantile(x, c(0.001, 0.999)),
     ylim = quantile(y, c(0.001, 0.999)),
     main = "log(wbcc) vs bicarbonate_max with LR fit")
abline(a = model$coefficients[1], b = model$coefficients[2],
       col = 'red', lwd = 4)
```



This is a decent fit. Next, we will examine the residuals of this model. Recall that the "model" data structure returned to us is a named list.

[31]:
```
print(names(model))
```

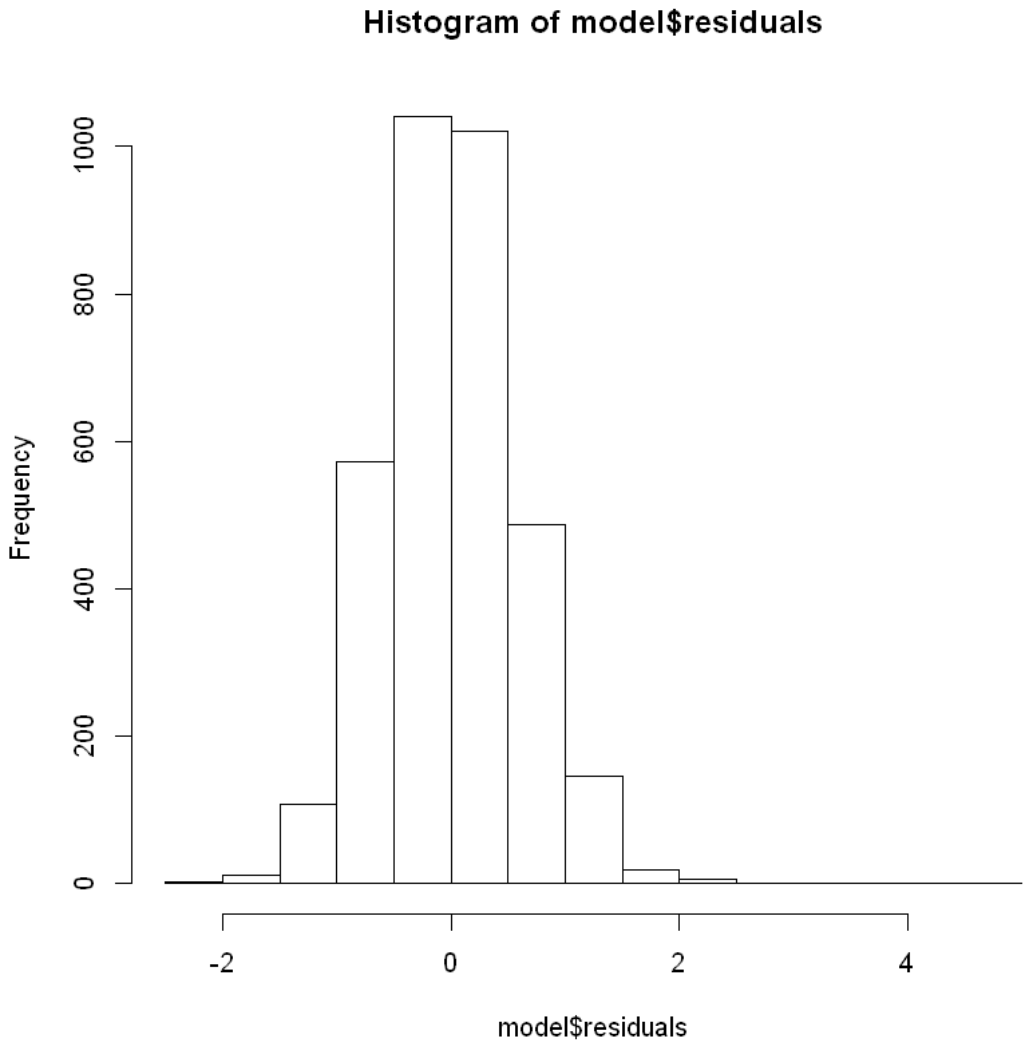
```
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
```

Further recall that to use a linear model, we had to assume that y was normally distributed around the mean that we predict at every level of x. We calculated the residuals manually in the categorical example earlier. However, to fit the model, R actually already calculated all the residuals and stores them in the "model" list.

We can extract them from the "model" list, and do a quick visualization of them by plotting a

histogram. Concretely, the residuals are the actual values of y minus their predicted means.

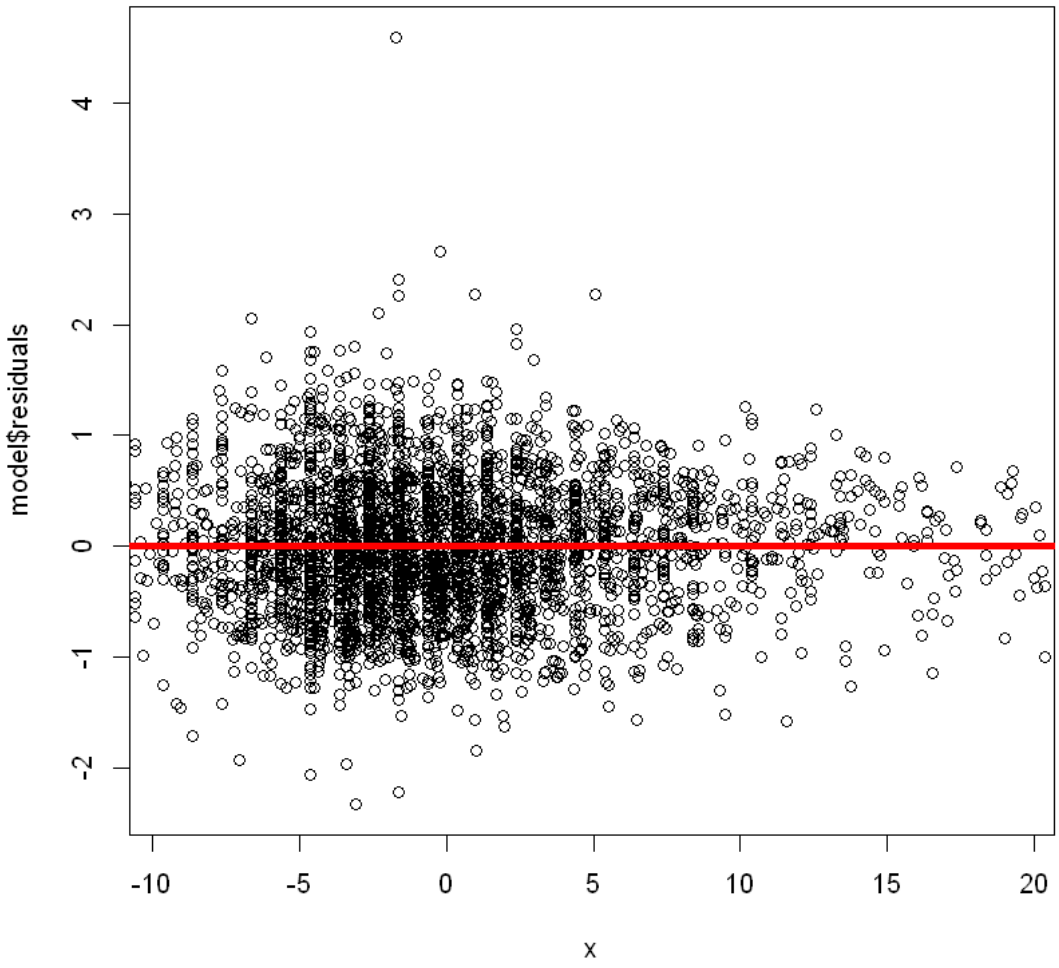
[32]:
```
hist(model$residuals)
```



Not perfect, but acceptable. Recall that this doesn't completely verify that the assumptions have been met, since the assumption is that the residuals need to be Normally distributed at *each setting* of x. This was easy in the categorical x case, since we could just subset and plot. As we discussed, this is much more difficult to verify in the continuous case, since at each potential level of x in your dataset you have either 0 or a few datapoints! We discussed binning, which extends the notion of "at x" to "near x", to make headway.

A slightly more advanced visualization we can do in the context of a continuous predictor is a **residual plot**. This is just a scatter plot with our residuals on the y-axis and our predicted data on the x-axis. This should look like normally distributed noise around the x-axis.

[33]:
```
plot(x, model$residuals,
     xlim = quantile(x, c(0.01, 0.99)))
abline(h = 0, col = 'red', lwd = 4)
```

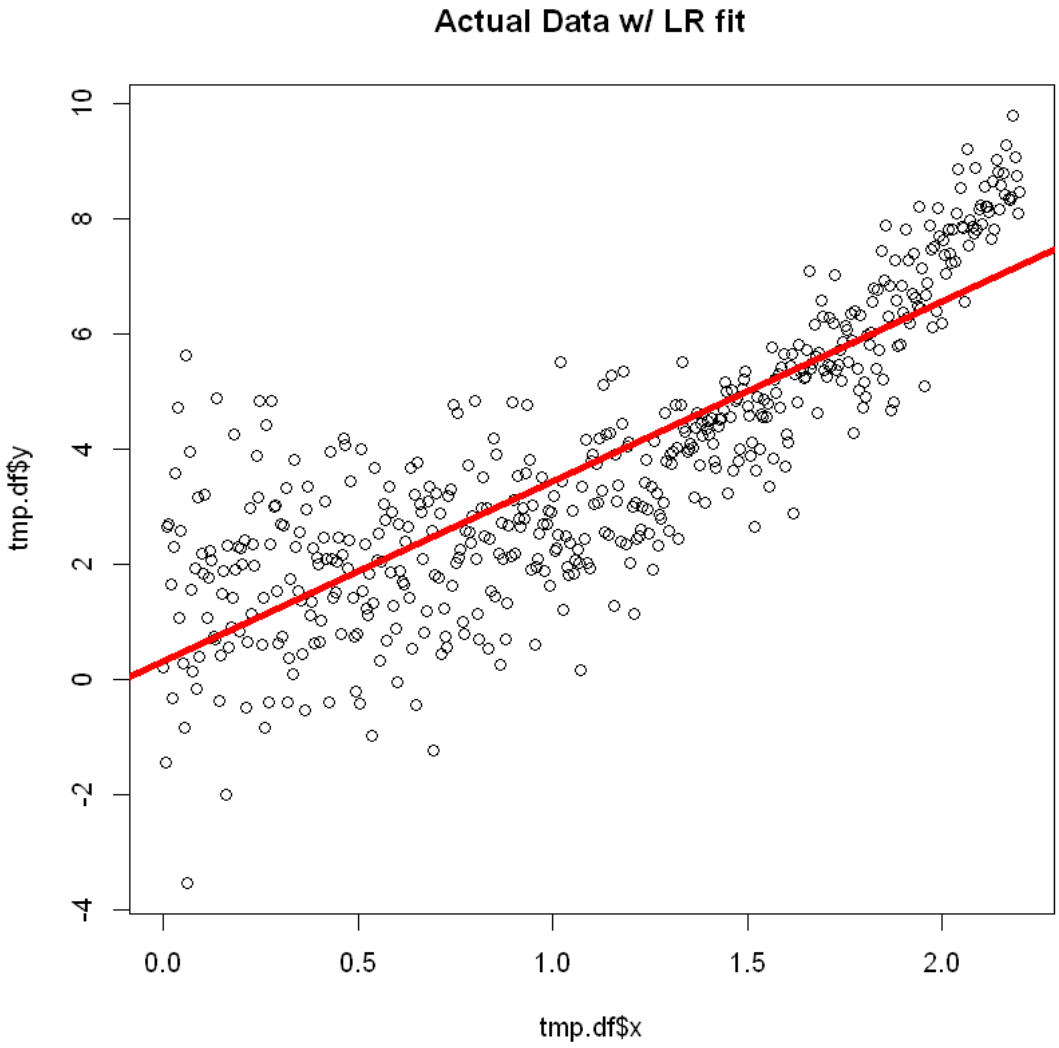


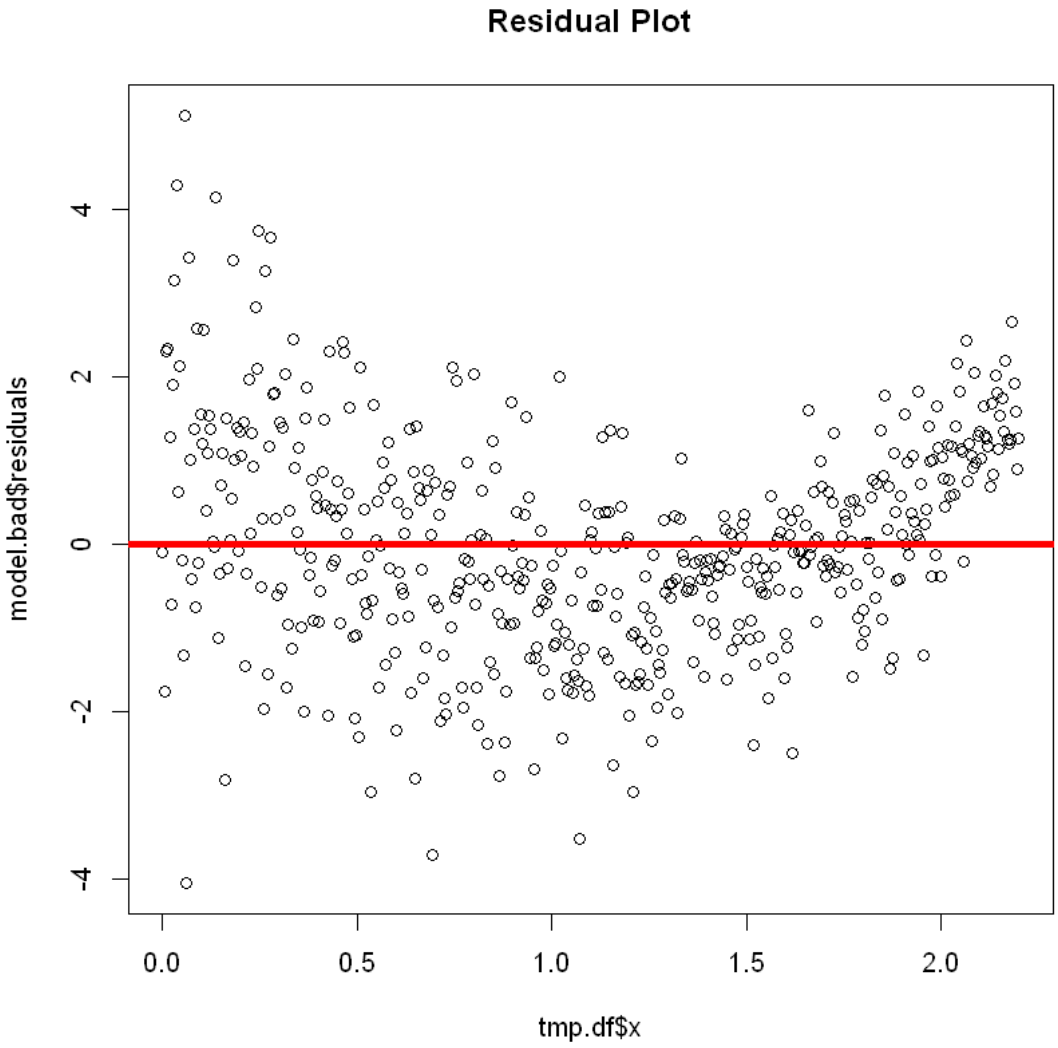
This is pretty good, so the model seems mostly appropriate.

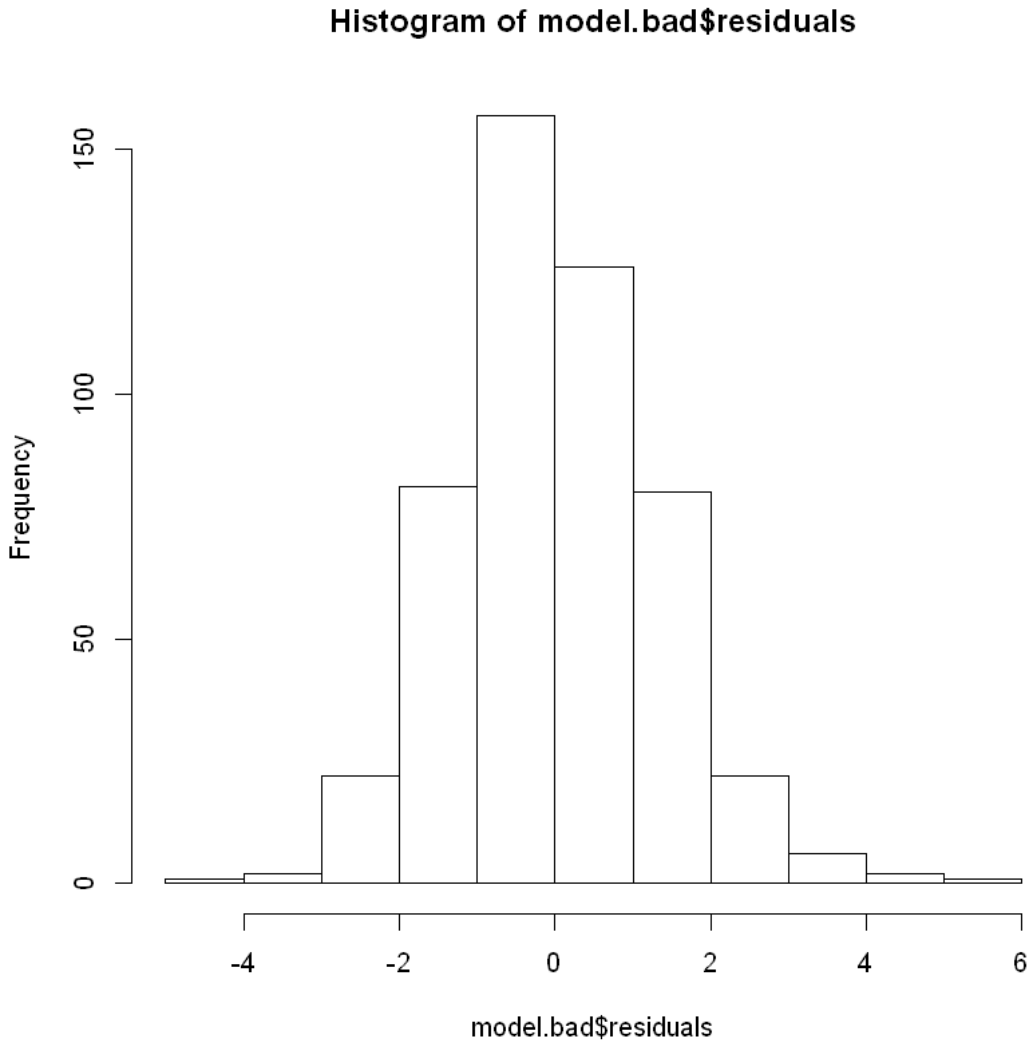
Let's look at an example where the linear model would be totally inappropriate. I will construct an example where the histogram of residuals are normal looking, but the assumptions are definitiely violated. The data follows an exponential trend, yet we will fit a simple linear regression to it.

[34]:
```
set.seed(2313) # for reproducibility

# simulate dataset
n <- 500
```

```r
x <- seq(0, 2.2, length.out = n)
y.mean <- exp(x)
residuals <- rnorm(n, 0, 2/(x+1))
y <- y.mean + residuals
tmp.df <- data.frame(x = x, y = y)

# fit model
model.bad <- lm(y ~ x, data = tmp.df)
# first plot
plot(tmp.df$x, tmp.df$y, main = 'Actual Data w/ LR fit')
abline(a = model.bad$coefficients[1], b = model.bad$coefficients[2],
       col = 'red', lwd=4)
# second plot
plot(tmp.df$x, model.bad$residuals, main = "Residual Plot")
abline(h = 0, col = 'red', lwd = 4)
# third plot
hist(model.bad$residuals, prob = F)
```


Actual Data w/ LR fit
tmp.df$y
tmp.df$x
10
8
6
4
2
0
-2
-4
0.0
0.5
1.0
1.5
2.0


Residual Plot
model.bad$residuals
4
2
0
-2
-4
0.0
0.5
1.0
1.5
2.0
tmp.df$x



If we just look at the histogram of residuals, we wouldn't see anything wrong! But as you can see from the residual plot, the distribution of residuals around any vertical slice is *not* always centered at 0. For example, on the right side of the plot, the errors are almost all positive, indicating the mean error is positive around those levels of x.

Another thing that needs to be checked is that the standard deviation of the residuals is the same *everywhere*. This is also not the case here, as the variability of the residuals is larger on the left side of the plot than the right.

We will not dive into model diagnostics in this session, but there are several visualization and calculation approaches that exist in practice for trying to figure out if your assumptions have been correctly met. I encourage you to read about them on your own. If you realize that the normally distributed errors assumption has not been met, there are some steps you can take that might fix the problem; we will discuss one set of approaches in the "basis expansion" section, and that

approach would indeed work for this particular example.

2.4.3 In-Class Exercise 3: Interpreting Your Model

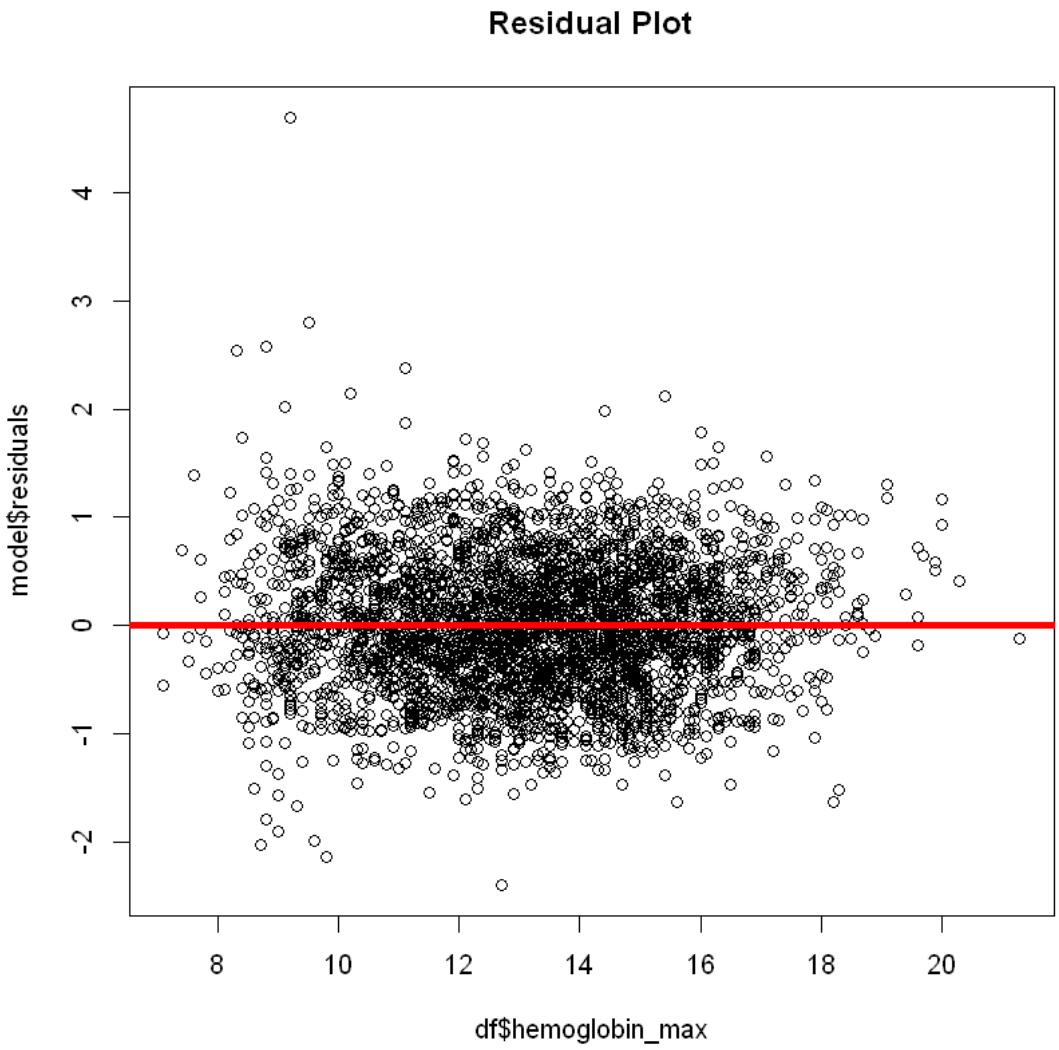
Fit a linear model where log(wbcc_max) is the dependent variable and hemoglobin_max is the independent variable. When doing this, choose whether to center and/or scale your independent variable based on the interpretation you want. Save the results to "model" and produce a summary.

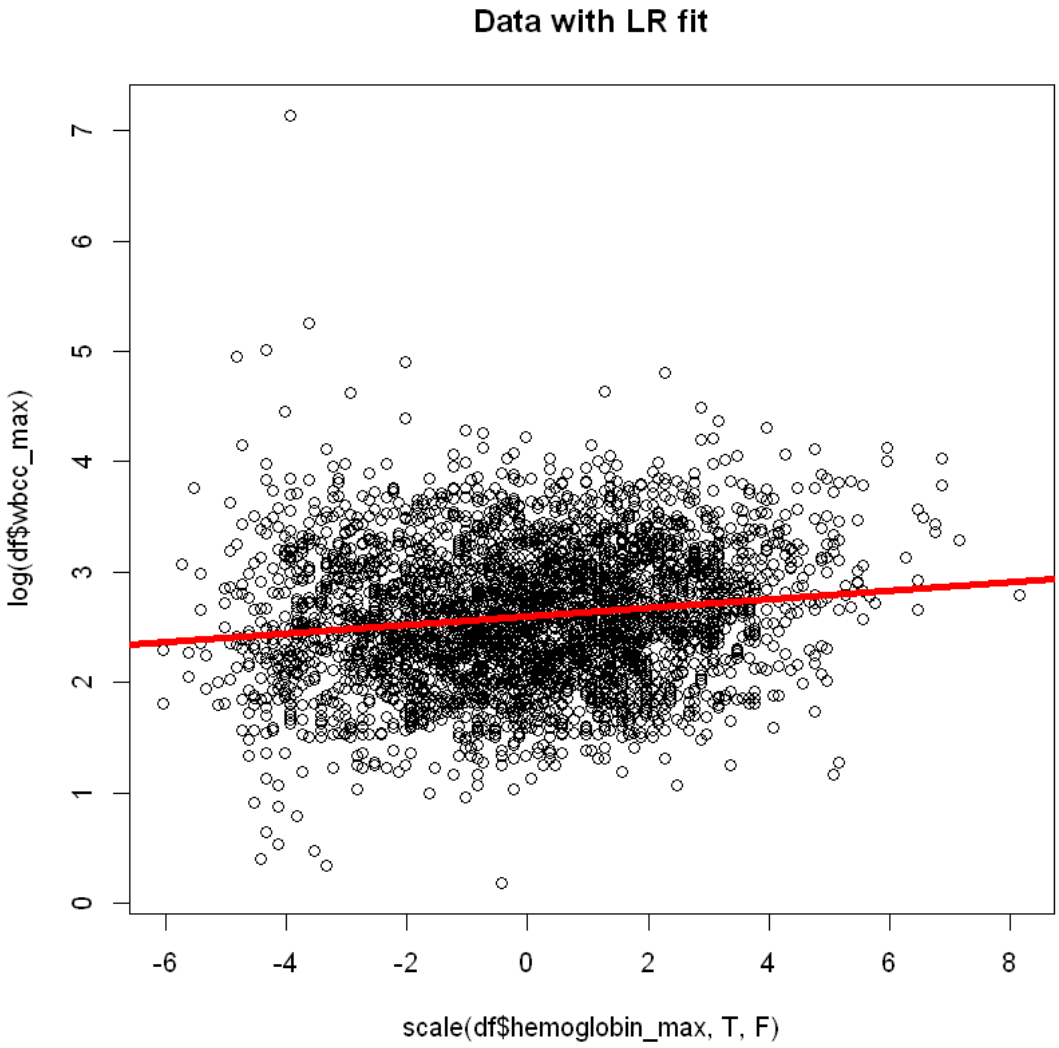
1. Although it is only a rough check for reasons we've discussed, make a residual plot (residuals located in model$residuals). Everything look kosher?
2. State an interpretation of the value of the intercept in a comment. Further comment on its associated p-value.
3. State an interpretation of the value of hemoglobin_max's coefficient in a comment. Further comment on its associated p-value.
4. Visualize your model fit using a scatterplot followed by the abline function. Recall your model coefficients are located in model\$coefficients. Does this make sense in the context of the R^2 value you saw in your model summary.

[35]:
```
model <- lm(log(wbcc_max) ~ scale(hemoglobin_max, T, F), data = df)

plot(df$hemoglobin_max, model$residuals,
    main = "Residual Plot") # check residuals
abline(h = 0, lwd = 4, col = 'red')

summary(model) # see coefficients

# visualize fit
plot(scale(df$hemoglobin_max, T, F), log(df$wbcc_max),
    main = "Data with LR fit")
abline(a = model$coefficients[1], b = model$coefficients[2],
       lwd = 4, col = "red")
```

```
Call:
lm(formula = log(wbcc_max) ~ scale(hemoglobin_max, T, F), data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-2.3980 -0.4249 -0.0300  0.4157  4.6916

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                 2.597050   0.010500 247.348   <2e-16 ***
scale(hemoglobin_max, T, F) 0.038506   0.004621   8.332   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6137 on 3414 degrees of freedom
Multiple R-squared:  0.01993,        Adjusted R-squared:  0.01964
```

```
F-statistic: 69.42 on 1 and 3414 DF,  p-value: < 2.2e-16
```





1. Yes, kosher.
2. The intercept is our estimate for the mean log(wbcc_max) for a person with an average hemoglobin_max (in this dataset). I use this interpretation, since I centered hemoglobin_max. The p-value tells us that there is extremely strong evidence that a linear model with a 0 intercept is non consistent with our data.
3. The coefficient for hemoglobin_max is our estimate for how much the mean log(wbcc_max) increases for every additional unit of hemoglobin above the average value. I use this interpretation, since I did not scale hemoglobin_max. The p-value tells us that there is extremely strong evidence that a linear model with a 0 coefficient for hemoglobin_max is non consistent with our data.

2.5 An Ordered Factor Predictor

How does R treat an ordered factored when used as the independent variable in a linear regression?

[36]:
```r
# make an ordinal variable for number of vaccinations: 0, 1, or 2+
df$num.vaccine.ord <- as.character(df$num_vaccine) # make a copy for␣
 ↪modification
df$num.vaccine.ord[df$num.vaccine >= 2] <- "2+"
df$num.vaccine.ord <- factor(df$num.vaccine.ord,
                             ordered = TRUE,
                             levels = c("0", "1", "2+"))

# run model
summary(lm(log(wbcc_max) ~ num.vaccine.ord, data = df))
```

```
Call:
lm(formula = log(wbcc_max) ~ num.vaccine.ord, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4788 -0.4311 -0.0198  0.4187  4.5601

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)        2.63268    0.01859 141.590   <2e-16 ***
num.vaccine.ord.L  0.05946    0.01913   3.108   0.0019 **
num.vaccine.ord.Q -0.03334    0.04133  -0.807   0.4199
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6189 on 3413 degrees of freedom
Multiple R-squared:  0.003329,        Adjusted R-squared:  0.002745
F-statistic:   5.7 on 2 and 3413 DF,  p-value: 0.003379
```

What does this L and Q stand for. Linear and Quadratic. R effectively performs a polynomial basis expansion on the under-the-hood integer coding for an ordered factor. You're likely reading this after I've already presented this Session, so go back and review polynomial expansions if needed. In this example, num.vacc.ord takes values "0", "1", and "2+", which correspond to integers 1, 2, and 3 respectively. R takes the number of levels in the ordered factor and does a polynomial basis expansion up to that number minus 1. Since we had 3 levels here, R did an expansion up till a quadratic (2).

Note that if you decided to do this polynomial basis expansion on the integer-code manually and check if you get the same results, you almost would. The p-values and test statistics would be the same. But the coefficients would be different. Why? You remember that contrast stuff from the categorical section? It has to do with that. R uses something called polynomial contrasts here, but we won't get into it.

So how to interpret the results? Coefficients aren't exactly interpetable. However, the p-values are. The p-value associated with L tells you if there is a linear trend with respect to the ordered levels. And the p-value associated with Q tells you if there is a quadratic trend. From our model

summary, we can see that there is only strong evidence of a linear trend.

3 Multi-Variate Linear Modeling

3.1 Adjustment

We are often interested in the relationship between a single independent and a dependent variable, which we will just call x_1 and y for short. However, nature is a complicated thing where millions of different variables are interacting with one another. In general, we don't have experimental control over many of these other factors; they all get to have their effects on y in whatever ways the data was created.

We call these variables **confounders**. Conceptually, confounders - may have a medium-strong effect on the outcome - are defined through the scientific question at hand; they are simply variables that we think have an association with the outcome, but we don't really care about - identify sources of variability in the outcome that we would like to remove when interpreting our main effect

As an example, lets say I have a binary treatment x_1 from a randomized control trial across several hospitals. I might identify the categorical variable of hospital as a source of variability in my outcome of interest. Ideally, I would like to estimate the effect of my treatment (read: coefficient of x_1) in a sense that really doesn't depend on a particular hospital. We would do this by dummy-coding hospitals and adding these variables to the model as confounders.

The language we use is that we try to *adjust* for confounding by including a series of potential confounding variables in our regression models. The choices for confounding variables need to be based on your scientific understanding of what you're trying to study, but very common choices in healthcare studies are demographics, comorbidities, and medication uses.

If you're interested in the relationship between x_1 and y, the coefficient associated with x_1 is going to be your primary target of intererest. Adjusting for other variables will very often take the unadjusted effect size for x_1 on y (read: coefficient of x_1 in non-adjusted regression) and move it closer to 0. However, this relationship does not always hold; it's possible for the adjusted effect size to grow larger.

Operationally, "adjusting" for other variables just amounts to throwing them into your regression model as extra independent variables. For demonstration, we're just going to run a regression with several independent variables of different data types and not worry about which independent variables we're interested in and which are confounders.

[37]:
```
continuous.indep.vars <- c("age", "hemoglobin_max", "plateletcount_max")
discrete.indep.vars <- c("race", "femalesex", "diabetescomp_elx_hospital",␣
 ↪"steroid")

# make formula
continuous.indep.vars <- paste0("scale(", continuous.indep.vars, ",T,F)") #␣
 ↪wrap each string in scale fxn
predictor.vars <- c(continuous.indep.vars, discrete.indep.vars) # combine all␣
 ↪predictors into one vector
predictor.string <- paste(predictor.vars, collapse = " + ") # merge all␣
 ↪predictors into one string separated by +'s
```

```
my.formula <- paste("log(wbcc_max)", predictor.string, sep = " ~ ") # prepend␣
 ↪outcome to string, separated by ~
my.formula

model <- lm(my.formula, data = df)
summary(model)
```

'log(wbcc_max) ~ scale(age,T,F) + scale(hemoglobin_max,T,F) + scale(plateletcount_max,T,F) + race + femalesex + diabetescomp_elx_hospital + steroid'

```
Call:
lm(formula = my.formula, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-2.1779 -0.3821 -0.0373  0.3331  4.7597

Coefficients:
                                Estimate Std. Error t value Pr(>|t|)
(Intercept)                    2.421e+00  2.160e-02 112.109  < 2e-16 ***
scale(age, T, F)               2.849e-03  6.377e-04   4.467 8.17e-06 ***
scale(hemoglobin_max, T, F)    2.062e-02  4.309e-03   4.785 1.78e-06 ***
scale(plateletcount_max, T, F) 1.504e-03  6.272e-05  23.970  < 2e-16 ***
raceAsian                     -4.274e-02  6.848e-02  -0.624   0.5326
raceBlack                     -5.290e-02  2.288e-02  -2.312   0.0208 *
raceOther                     -4.149e-02  2.902e-02  -1.430   0.1529
femalesex                     -8.955e-02  1.924e-02  -4.654 3.38e-06 ***
diabetescomp_elx_hospital      9.639e-02  1.948e-02   4.948 7.86e-07 ***
steroid                        2.928e-01  2.022e-02  14.479  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5449 on 3406 degrees of freedom
Multiple R-squared:  0.2292,        Adjusted R-squared:  0.2272
F-statistic: 112.5 on 9 and 3406 DF,  p-value: < 2.2e-16
```

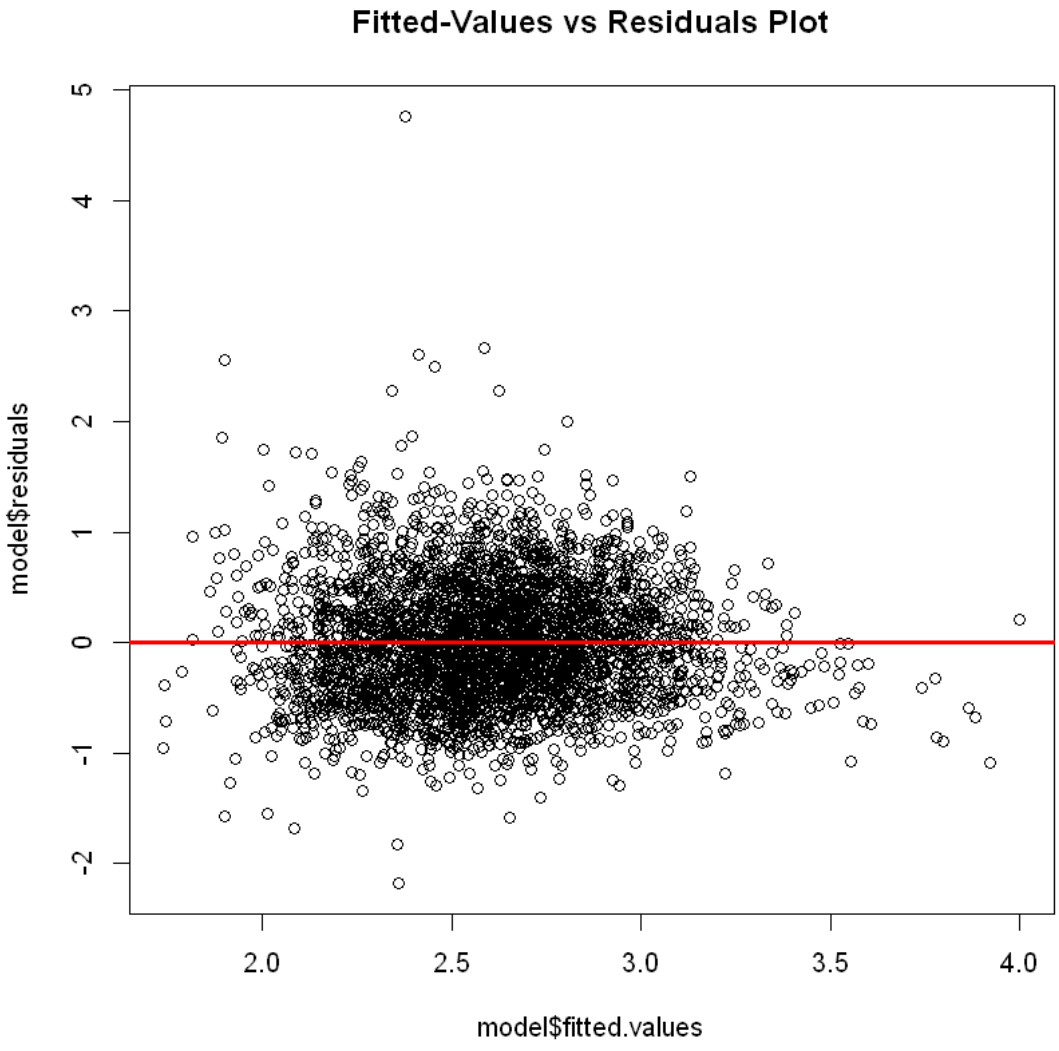
So for this regression, we interpret the intercept as predicted mean log(wbcc_max) when someone is male, white, non-diabetic, not on a steroid, with an average age, average hemoglobin_max, and average plateletcount_max. Furthermore, several coefficients we had looked at before are now different. Each coefficient associated with a variable is to be interpreted as the change in mean-outcome for every unit increase of that variable.

Note that as you add more variables to your regression, your standard errors (and thus confidence intervals) will usually increase. With a fixed size dataset, it should feel intuitive that it is harder to estimate many things than a few things.

3.2 Fitted-Values vs Residuals Plots

Once we start adjusting for multiple things, it would be almost impossible to visualize the residual plot, since it would be taking place in a high-dimensional space. We can make a compromise in accuracy and instead look at the fitted values vs residuals plot. This is almost exactly like the residual plot, except the x-axis now consists of our $\hat{y}$ predictions. This keeps the plot 2-D. We lose some information with respect to checking the assumption, since it is possible that multiple different levels of x could lead to the same fitted value $\hat{y}$. But this is a reasonable compromise to make the assumption checking tractable. Let's produce a plot for the previous model.

[38]:
```
plot(model$fitted.values, model$residuals,
    main = "Fitted-Values vs Residuals Plot")
abline(h = 0, lwd = 3, col = 'red')
```



Seems to be pretty good! Maybe the residuals are a little unbalanced on the right, but the sample size in that region is too small to be sure. I'd call this good enough.

3.2.1 In-Class Exercise 4: Getting Adjusted

You are interested in modeling the association between log(wbcc_max) (y) and creatinine_max (x_1). Let's adjust for age, femalesex, steroid, and consult_nephrology.

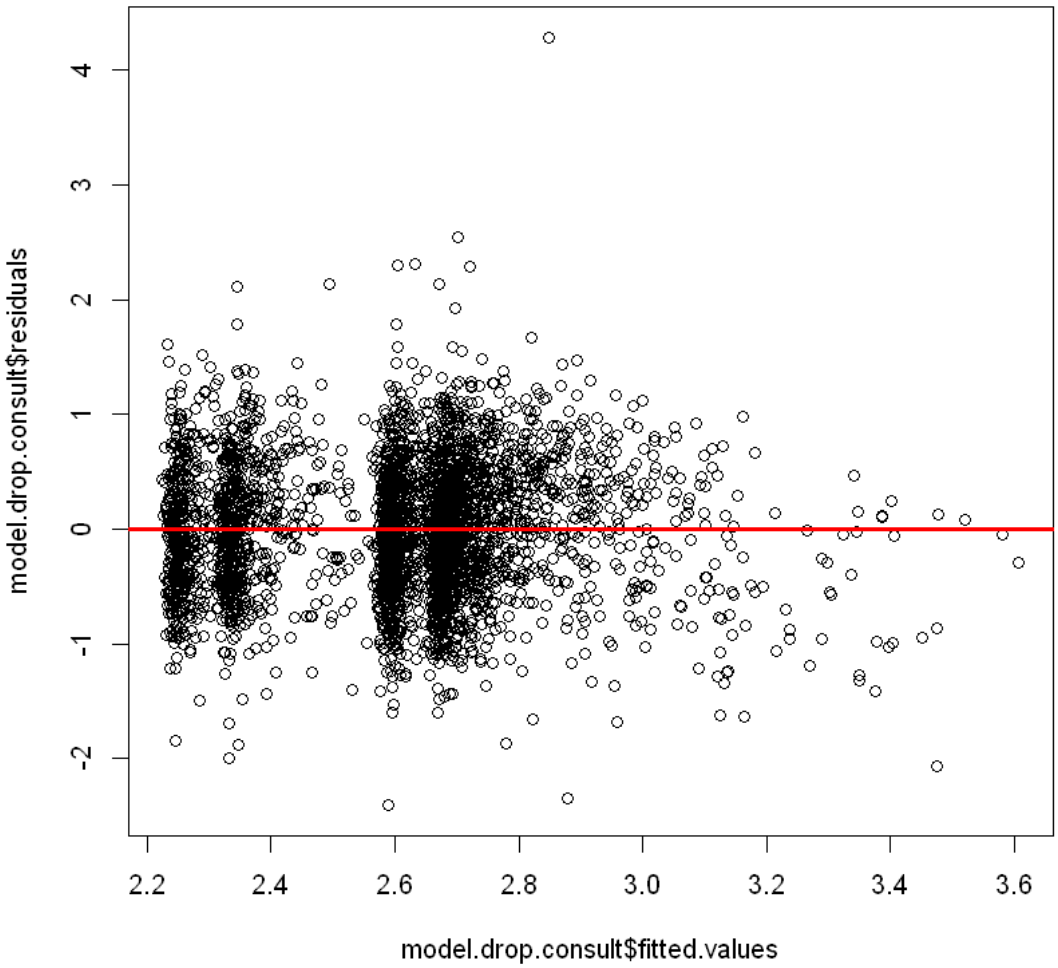
1. Run the linear model twice, once with adjustment and once without. Your inferential target is your coefficient for creatinine_max. Did the coefficient change between the two models? State in a comment why you think this is.
2. If any of your coefficients were insignificant, try dropping them and running the analysis again. Did this change much? Do you think this approach is statistically acceptable or not?
3. Someone is quite likely to get a nephrology consult when their kidney is injured, meaning they have a high creatinine usually. If we're trying to isolate the association of the outcome with creatinine_max, does it make sense to adjust for consult_nephrology? Run the analysis without consult_nephrology and comment on what happens to the coefficient for creatinine_max and why.
4. Compute a fitted-values vs residual plot to assess the appropriateness of this final model.

[39]:
```
model.smol <- lm(log(wbcc_max) ~ scale(creatinine_max,T,F), data = df)
summary(model.smol)
print('--------------------------------------------------------------------------------')
model <- lm(log(wbcc_max) ~ scale(creatinine_max,T,F) + scale(age,T,F) +
            femalesex + steroid + consult_nephrology, data = df)
summary(model)
print('--------------------------------------------------------------------------------')
model.drop.age <- lm(log(wbcc_max) ~ scale(creatinine_max,T,F) +
            femalesex + steroid + consult_nephrology, data = df)
summary(model.drop.age)
print('--------------------------------------------------------------------------------')
model.drop.consult <- lm(log(wbcc_max) ~ scale(creatinine_max,T,F) +
            scale(age,T,F) + femalesex + steroid, data = df)
summary(model.drop.consult)
plot(model.drop.consult$fitted.values, model.drop.consult$residuals)
abline(h = 0, lwd = 3, col = 'red')
```

```
Call:
lm(formula = log(wbcc_max) ~ scale(creatinine_max, T, F), data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-2.3307 -0.4150  0.0008  0.4231  4.4143

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)                2.597050   0.010330  251.41   <2e-16 ***
scale(creatinine_max, T, F) 0.049962   0.003677   13.59   <2e-16 ***
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6038 on 3414 degrees of freedom
Multiple R-squared:  0.0513,        Adjusted R-squared:  0.05102
F-statistic: 184.6 on 1 and 3414 DF,  p-value: < 2.2e-16


[1]
"------------------------------------------------------------------------"


Call:
lm(formula = log(wbcc_max) ~ scale(creatinine_max, T, F) + scale(age,
    T, F) + femalesex + steroid + consult_nephrology, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-2.5362 -0.3971 -0.0049  0.3909  4.0824

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                 2.337e+00  2.019e-02 115.742  < 2e-16 ***
scale(creatinine_max, T, F)  2.038e-02  4.013e-03   5.080 3.98e-07 ***
scale(age, T, F)            -3.602e-05  6.178e-04  -0.058 0.953507
femalesex                   -7.513e-02  1.981e-02  -3.792 0.000152 ***
steroid                      3.251e-01  2.094e-02  15.524  < 2e-16 ***
consult_nephrology           3.421e-01  2.740e-02  12.484  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5683 on 3410 degrees of freedom
Multiple R-squared:  0.1604,        Adjusted R-squared:  0.1592
F-statistic: 130.3 on 5 and 3410 DF,  p-value: < 2.2e-16


[1] "------------------------------------------------------------------------"


Call:
lm(formula = log(wbcc_max) ~ scale(creatinine_max, T, F) + femalesex +
    steroid + consult_nephrology, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-2.5365 -0.3972 -0.0044  0.3915  4.0818

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                 2.336719   0.020174 115.825  < 2e-16 ***
```

```
scale(creatinine_max, T, F)  0.020375   0.004009   5.082 3.94e-07 ***
femalesex                   -0.075236   0.019716  -3.816 0.000138 ***
steroid                      0.325102   0.020939  15.526  < 2e-16 ***
consult_nephrology           0.342170   0.027381  12.497  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5682 on 3411 degrees of freedom
Multiple R-squared:  0.1604,        Adjusted R-squared:  0.1594
F-statistic: 162.9 on 4 and 3411 DF,  p-value: < 2.2e-16


[1] "--------------------------------------------------------------------------"


Call:
lm(formula = log(wbcc_max) ~ scale(creatinine_max, T, F) + scale(age,
    T, F) + femalesex + steroid, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4068 -0.4086 -0.0107  0.3910  4.2877

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                 2.3984410  0.0200125 119.847  < 2e-16 ***
scale(creatinine_max, T, F) 0.0449359  0.0035761  12.566  < 2e-16 ***
scale(age, T, F)           -0.0003250  0.0006312  -0.515 0.606694
femalesex                  -0.0761105  0.0202536  -3.758 0.000174 ***
steroid                     0.3422848  0.0213659  16.020  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5811 on 3411 degrees of freedom
Multiple R-squared:  0.122,        Adjusted R-squared:  0.121
F-statistic: 118.5 on 4 and 3411 DF,  p-value: < 2.2e-16
```



1. A lot of the variation in log(wbcc_max) can be attributed to other variables other than creatinine. For example, the R^2 value in the unadjusted model was around 5%, where it was around 16% in the adjusted model. Some of these variables overlap with creatinine in their ability to explain the variation in the outcome, so when adjusting, creatinine's coefficient gets reduced by over half. This is the usual pattern we see, though it is possible for the coefficient increase after adjustment in some specific scenarios.
2. This barely changed the results of the analysis. However, this procedure might in larger models with more variables. If you have a bunch of insignificant variables and drop them and rerun, you're essentially performing multiple testing and not reporting it. This can bias the estimates left in your regression model towards being too significant. In general, if your goal is inference on a particular coefficient, you should leave insignificant predictors in your model. You chose the variables to include in your regression for a scientific reason; if they aren't significant, that's an interesting finding.

3. Because of the extremely strong correlation between creatinine_max and consult_nephrology, the latter was "stealing" away some of the effect size from creatinine. Since our goal is to isolate the association with creatinine, we should not be adjusting for this variable in the first place. Removing it doubles the coefficient for creatinine, but not quite to the level of the unadjusted model.
4. Looks good. Maybe a little unbalanced on the right, but the sample size is too small to be sure.

3.3 Flexible Modeling

Let's say you think your linear model isn't flexible enough to model your data in the way you want it to. Or maybe you're not getting normally distributed residuals and are trying to fix it. Or maybe there's some kind of hypothesis you're trying to test that can't be captured in a vanilla linear model. What to do? Well we can use the variables we already have to construct new variables in clever ways.

3.3.1 Feature Engineering

Feature engineering refers to the art of creating new variables that might be more informative than the features you were originally given. How to do this is extremely domain-specific, but I shall give you two examples.

Sometimes, we already have domain knowledge that lets us combine predictors or "features" in meaningful ways. For example, BMI is an engineered feature from height and weight. Estimated glomular filtration rate can be a useful engineered feature of kidney function, which is derived from age, race, sex, and serum creatinine. An estimate for mean arterial blood pressure can be derived from 2/3 diastolic + 1/3 systolic.

There is one note of caution: if your engineered feature is a linear combination of other features *and* you include those other features in your model, you will get an error for mathematical reasons. You cannot have systolic, diastolic, and mean arterial pressure all in your model; you must pick only 1 or 2 of these to keep.

Another example of feature engineering is grouping categorical/binary variables together. If you have 10 drugs for hypertension in your dataset, then there's a good chance some of these are pretty rarely administered, and your estimates based on them will be poor due to effective sample size issues. It might be a good idea to engineer a new variable that just indicates whether the patient was exposed to *any* hypertensive medication. Another situation might be when a categorical variable has a lot of levels. You might choose to coarsen that variable into a smaller number of meaningful levels. For example, when creating this dataset, I coarsened the race variable to 4 levels from the 10-15 levels that were initially present.

3.3.2 Basis Expansions

Basis expansions refer to the act of "expanding" one (or more) variable into a larger number of variables through the use of pre-set transformations/combinations.

Interactions An interaction is a specific type of expansion of the following type:

$$(1, x_1, x_2) \longrightarrow (1, x_1, x_2, x_1 * x_2)$$

Let's walk through an example that will demonstrate why you might want to do this.

Suppose that you are interested in learning about the relationship between log(wbcc_max) (y) and steroid use (x_s). You think that female vs male (x_f) might be a relevant confounding variable. So you adjust for it and choose the following model:

$$\mu_y(x_s, x_f) = \beta_0 + \beta_1 x_s + \beta_2 x_f$$

When we have a male, i.e. $x_f = 0$, the equation simplifies to

$$\mu_y(x_s, 0) = \beta_0 + \beta_1 x_s$$

When we have a female, i.e. $x_f = 1$, the equation looks like

$$\mu_y(x_s, 1) = (\beta_0 + \beta_2) + \beta_1 x_s = \tilde{\beta}_0 + \beta_1 x_s$$

Being female corresponds to just changing the intercept compared to being male. Here is the model run

[40]:
```
lm(log(wbcc_max) ~ steroid + femalesex, data = df)
```

```
Call:
lm(formula = log(wbcc_max) ~ steroid + femalesex, data = df)

Coefficients:
(Intercept)      steroid    femalesex
     2.4040       0.3567      -0.1101
```

Now instead, what if we wanted to try the following model.

$$\mu_y(x_s, x_f) = \beta_0 + \beta_1 x_s + \beta_2 x_f + \beta_3 x_s * x_f$$

What effects might this have?

When we have a male, i.e. $x_f = 0$, the equation looks the same as before:

$$\mu_y(x_s, 0) = \beta_0 + \beta_1 x_s$$

When we have a female, i.e. $x_f = 1$, the equation now looks like

$$\mu_y(x_s, 1) = (\beta_0 + \beta_2) + (\beta_1 + \beta_3) x_s = \tilde{\beta}_0 + \tilde{\beta}_1 x_s$$

As you can see, being female now changes both the intercept *and* the coefficient for steroid compared to the male equation. This means that males and females now each get their own coefficient for steroid, rather than having to share a common one. The equation also shows us how to combine the coefficients to derive these. The syntax for running this model is:

[41]:
```
summary(lm(log(wbcc_max) ~ steroid*femalesex, data = df))
```

```
Call:
lm(formula = log(wbcc_max) ~ steroid * femalesex, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4736 -0.4243 -0.0241  0.4048  4.3807

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)        2.41366    0.02483  97.226  < 2e-16 ***
steroid            0.34276    0.02979  11.504  < 2e-16 ***
femalesex         -0.13047    0.03603  -3.621 0.000297 ***
steroid:femalesex  0.03001    0.04375   0.686 0.492856
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5943 on 3412 degrees of freedom
Multiple R-squared:  0.08149,        Adjusted R-squared:  0.08068
F-statistic: 100.9 on 3 and 3412 DF,  p-value: < 2.2e-16
```

The male coefficient for steroid is 0.34276, but the female coefficient is $(0.34276 - 0.03001)$. Notice that we still let femalesex have its own effect on the outcome, separately from how it interacts with steroid. This same type of reasoning goes through if we replaced steroid with some other continuous variable. The interaction between any variable and a categorical variable is quite useful in medicine for the purposes of interpretation and testing hypotheses.

Suppose that our primary inferential target was the coefficient associated with steroid. Testing if the interaction coefficient between steroid and sex is significantly different than 0 can help us decide answer the scientific question of if there really is a differential effect of steroids between men and women (after potentially adjusting for a bunch of other things).

When interacting two continuous variables, this sort of intuition breaks down a little. In this case, the x's are not just 0/1, so the coefficients are always "active", so we can't use the same reasoning of interpretation. The product between two continuous variables just serves to create a new type of variable that might help model the data in a more flexible way. In fact, interacting a continuous variable with itself (possibly several times) is called a *polynomial basis expansion*.

Polynomial Expansions So maybe you think the relationship between two variables isn't actually linear. Maybe its quadratic. Or Cubic? Well we can actually model these types of relationships by what's known as a polynomial basis expansion. Suppose we start with this model.

$$\mu_y = \beta_0 + \beta_1 x$$

We could "interact" a variable with itself to give us the following expansion

$$(1, x) \longrightarrow (1, x, x * x, x * x * x) = (1, x, x^2, x^3)$$

This leads to the following model:

$$\mu_y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

Is this still a linear model? It is. A linear model means that the equation is linear *with respect to the coefficients*, not with respect to the variables.

If your original model has 2 continuous variables that you think might have a nonlinear relationship with the data, try creating a quadratic basis expansion:

$$(1, x_1, x_2) \longrightarrow (1, x_1, x_2, x_1^2, x_2^2, x_1 * x_2)$$

This leads to the following linear model:

$$\mu_y = \beta_0 + \beta_1 x_a + \beta_2 x_b + \beta_3 x_a^2 + \beta_4 x_b^2 + \beta_5 x_a * x_b$$

Exponential/Logarithmic Expansions There are many other types of basis expansions you can perform. Here is an example. If y seems to be exponentially increasing in x, you could try adding an exponential term in your expansion like follows:

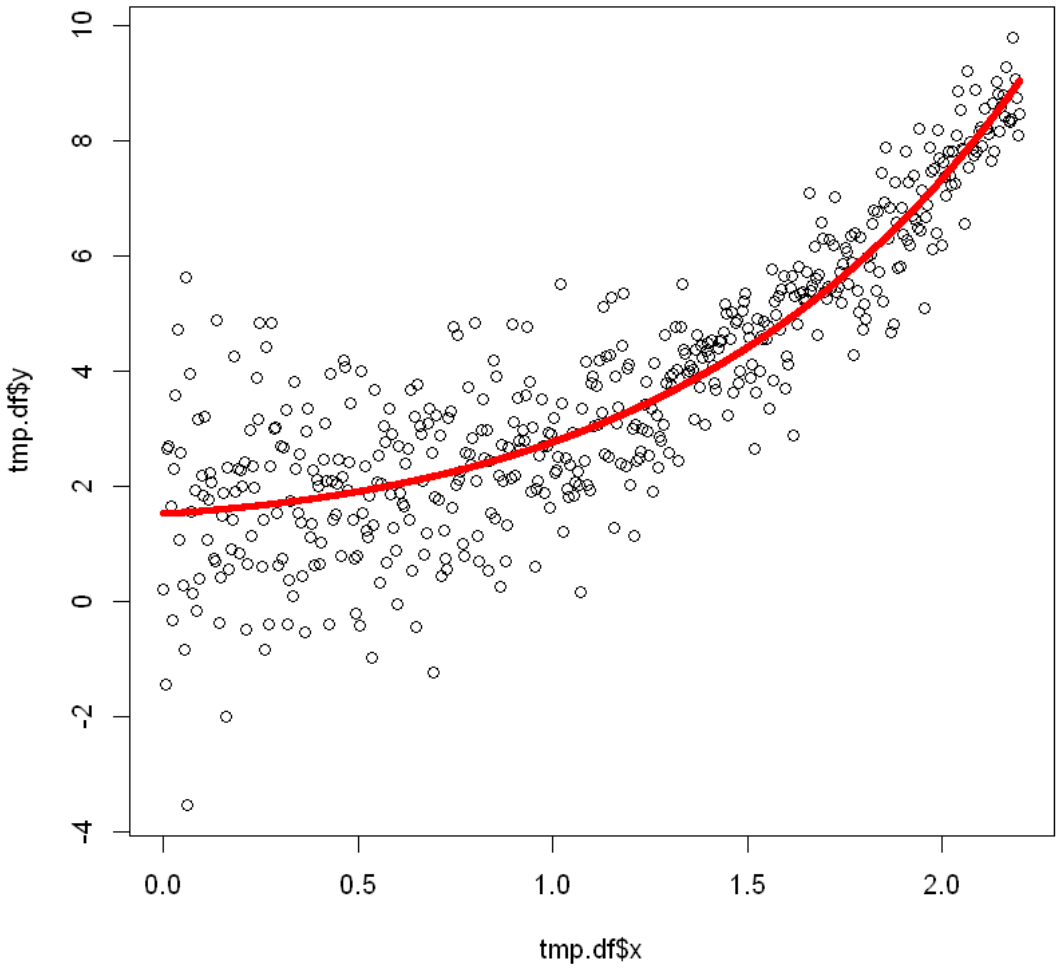
$$(1, x) \longrightarrow (1, x, \exp(x))$$

which leads to the following model:

$$\mu_y = \beta_0 + \beta_1 x + \beta_2 \exp(x)$$

Let's look at the badly behaved model that I created to show a violation of the linear assumption. This time, we will model it using the above equation and show that this fixes one of the two problems we identified; it still does not fix the residual standard deviation varying across the plot.

[42]:
```
set.seed(2313)
n <- 500
x <- seq(0, 2.2, length.out = n)
residuals <- rnorm(n, 0, 2/(x+1))
tmp.df <- data.frame(x = x, y = exp(x) + residuals)
model <- lm(y ~ x + exp(x), data = tmp.df)
plot(tmp.df$x, tmp.df$y)

grid <- seq(0, 2.2, length.out = 5000)
y.hat <- model$coefficients[1] + model$coefficients[2]*grid +␣
 ↪model$coefficients[3]*exp(grid)
lines(grid, y.hat, col = 'red', lwd=4)
```

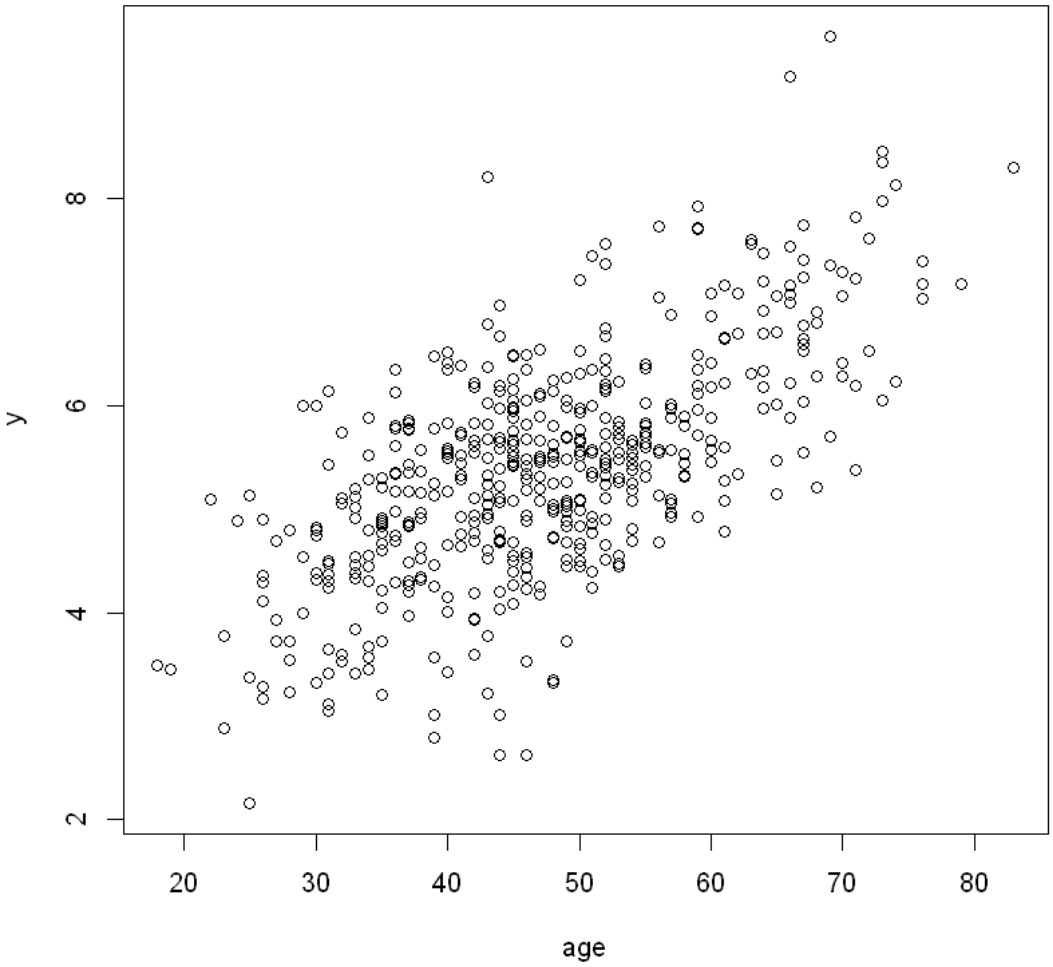


Again, this is still a linear model. In fact, you can use any pre-determined functions (like exp, log, sin, cos, etc…) to create new terms to add to your model, and it stays linear! We'll choose a very specific type of pre-determined function in the next example to give us a nice interpretation.

Piece-Wise Linear Expansions Maybe you believe that age has a set linear relationship to some outcome of interest, but up to a point. Maybe you believe that linear relationship changes around the age of 60, and you'd like to scientifically test this hypothesis. A naive way of going about this would be to split into subgroups, those below 60 and above. Then you might compare means. But we can do a more sophisticated analysis with linear models. Let's make up some data to work with.

[43]:
```
set.seed(216113)
n <- 500
```

```
age <- rnbinom(n, size = 20, prob = 0.3)
cutoff <- 60
indicator <- (age >= cutoff)
expansion.var <- indicator*(age - cutoff)

betas <- c(3, 0.05, 0.04)
mu.y <- betas[1] + betas[2]*age + betas[3]*expansion.var
y <- mu.y + rnorm(n, mean = 0, sd = 0.8)
plot(age, y)
```



We wanted to test if people above 60 have a different trend to their linear relationship than everyone else. We can make a linear model representing this explicit using a basis expansion. In terms of the how to think about this, you first make an indicator variable for if the patient's is older than 60 or

not. This determines when the kink in the linear function should turn "on". Then you interact this with the number of years a patient is older than 60. This ensures that at the value $x = 60$, this new term is 0. This setup ensures continuity in your piece-wise linear function, which is probably what you want. This expression is actually equivalent to taking the maximum between $(x - 60)$ and 0. In algebra, we are claiming

$$\mathbf{1}\{x >= 60\} * (x - 60) = \max(0, x - 60)$$

Using this, our particular expansion will look like

$$(1, x) \longrightarrow (1, x, \max(0, x - 60))$$

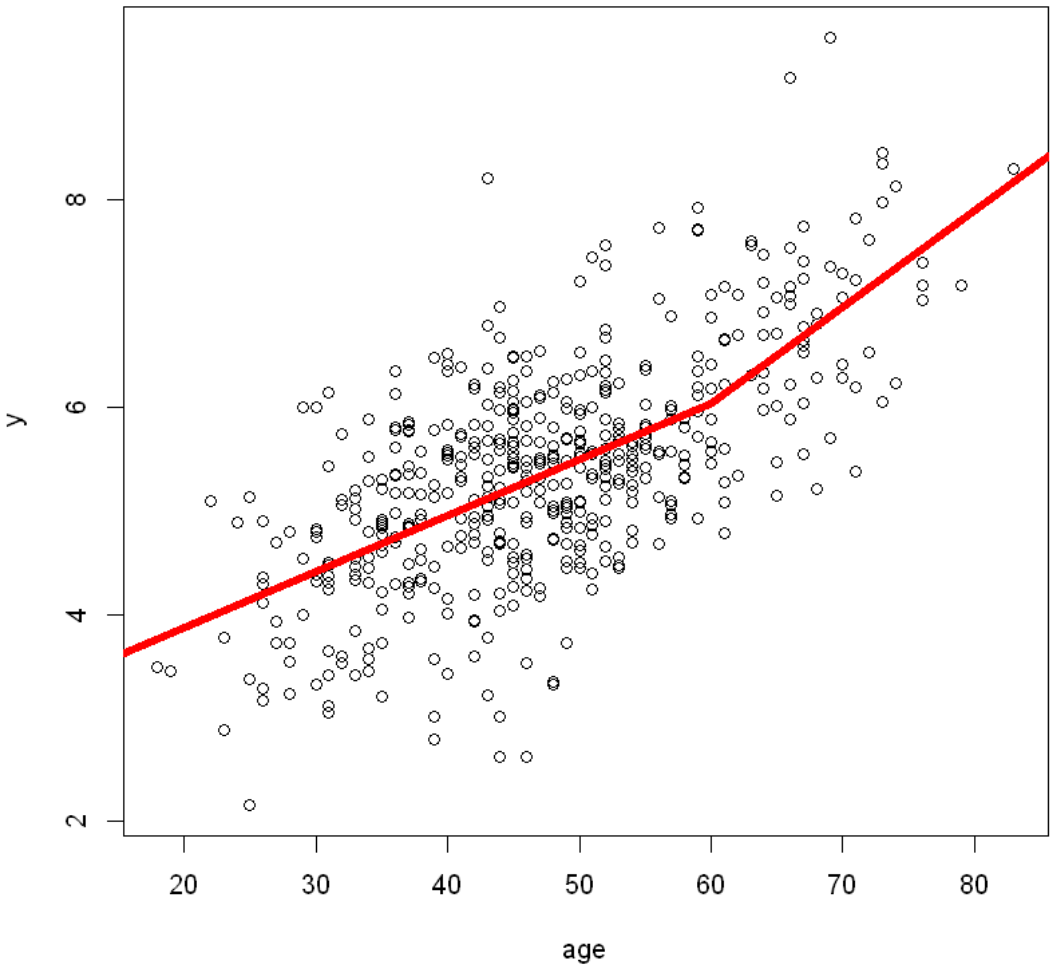
which leads to the following model:

$$\mu_y = \beta_0 + \beta_1 x + \beta_2 \max(0, x - 60)$$

Then $\hat{\beta}_1$ describes the linear trend for those under age 60. And $\hat{\beta}_1 + \hat{\beta}_2$ describes the trend for those over age 60. Let's run the model and plot to verify.

[44]:
```
age.expansion.term <- (age >= 60)*(age-60)
model <- lm(y ~ age + age.expansion.term)
summary(model)

plot(age, y)
grid <- seq(15, 90, length.out = 5000)
grid.transformed <- (grid >= 60)*(grid - 60)
y.hat <- model$coefficients[1] + model$coefficients[2]*grid +␣
 ↪model$coefficients[3]*grid.transformed
lines(grid, y.hat, col = 'red', lwd=4)
```

```
Call:
lm(formula = y ~ age + age.expansion.term)

Residuals:
     Min       1Q   Median       3Q      Max
-2.65213 -0.50786  0.00497  0.53406  3.09692

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)        2.787900   0.189516  14.711   <2e-16 ***
age                0.054155   0.004147  13.058   <2e-16 ***
age.expansion.term 0.039092   0.015609   2.504   0.0126 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8184 on 497 degrees of freedom
Multiple R-squared:  0.4389,        Adjusted R-squared:  0.4367
F-statistic: 194.4 on 2 and 497 DF,  p-value: < 2.2e-16
```



As we can see, the coefficient associated with the new term is significant, which supports the conclusion that the kink in the function above represents a real effect.

4 Relaxing Assumptions

The estimates that we get from R when we use the lm function are called the OLS estimates, short for ordinary least squares. OLS is an optimal estimator corresponding to the basic linear model we described. In each header, we give a number of ways the basic linear model corresponding to the OLS estimator may be violated in practice; this is followed by some known extensions of the linear model or fixes.

4.0.1 Non-Normal Residuals

This is the easiest assumption to fix. If you have enough data, it doesn't matter! The CLT kicks in and OLS works correctly. However, residuals must always be centered at 0 for every level of x.

One note of caution - what it means for your data to be "large enough" can be very different in a regression context. The more predictors you have, the more data you need to be "large enough", and this relationship can grow exponentially. So best practice is to try to keep your models relatively parsimonious if you're trying to do inference over coefficients and not dealing with big data.

4.0.2 Heteroskedasticity

Heteroskedasticity refers to when the basic linear model is violated due to there not being a common standard deviation for every level of x. Different levels of x have different amounts of variation.

The first method to fix this is to apply some *variance-stabilizing transformation* to y to make the standard deviation look more uniform across levels of x. However, this can mess up the units of y and make things less interpretable.

A second solution is to just use OLS as usual. In this case, our coefficient estimates are still correct in the sense that they are *unbiased*, i.e. their distributions are correctly centered around the true values. However, your standard errors, p-values, etc… will be wrong. We fix this by using *robust* aka "Huber-White" standard error formulas, which are available in packages. These will inflate your standard errors from the values the lm function gives you and will typically be overly conservative.

A third and best solution is to directly use a probability model that allows for heteroskedasticity. WLS, short for weighted least squares, is an estimator for the heteroskedastic linear model. If you want to accomodate heteroskedasticity in this way, talk to your local statistician about using two-stage WLS.

4.0.3 Model Misspecification

Let's say you have evidence that the form of your linear equation is wrong, maybe through plotting or something. Something like the example I gave before where the data had an exponential trend, but I fit it with a straight line. You are no longer allowed to interpret your coefficients as the effect of an independent variable on the true mean; now, you are only estimating the coefficient associated with the best possible linear equation that could fit the data. This is a much weaker claim.

You can still get correct confidence intervals for this by using robust standard errors, like described in the previous section.

Typically, try to avoid model misspecification if you can by adding new variables to your model or using some of the basis expansion techniques that we've talked about.

4.0.4 Different Outcome Types

Suppose you have binary outcome data. Or count outcome data? Or time-to-event data? How might we extend the linear model to account for these? This will be the goal of the next session, so more to come! We will discuss generalized linear models and linear survival models.

4.0.5 Non-independence

The rows in your data can be correlated through various reasons. Two common reasons are time-series data or more generally clustered data, such as when you have multiple data points within a classroom and your dataset contains multiple classrooms' data.

For time-series, there are a hierarchy of linear models that have been developed. Auto-regressive (AR) and moving-average (MA) models are two basic types of linear models. These can be combined to form ARMA models. These can be extended to form ARIMA. The extensions continue in many directions.

For more generally clustered data, we typically rely on linear mixed-effects (LME) models or generalized estimating equation (GEE) models. I prefer the latter for its robustness: data can be of various outcome types, residuals can have an arbitrary distribution as long the mean is 0, correlation within a cluster can be arbitrary, and model fitting is computationally fast. Note that if you have many related time-series (like admission-to-discharge data for many patients in the hospital), you might use this kind of approach rather than traditional time-series models.