Chapter 5 and 6

DJM

21 February 2017

Exam admonitions

0. You must compile the `.Rmd` to `.html`
1. No code in the `html`.
2. If you put plots or tables in, you must talk about them.

   Rule of thumb: if you don't have anything good to say about a number, don't give the number (or plot) at all.
3. **MOST IMPORTANT** you must explain your results. Simply providing them is not likely to get you credit.
4. Look over the model solutions for the last assignments.

Project progress report

- Due **8 March at 11:59 pm** (just over 2 weeks from today)
- Your report should have 3 components:
  1. A list of teammate names and an explanation of what is interesting to you about this data.
  2. A short introductory paragraph introducing the data and describing some potential questions you might investigate.
  3. A lengthy exploratory data analysis.
- The third part is a big deal.
- You need to provide evidence that you have explored the data carefully and meaningfully.
- Code must be integrated.
- Think of this like HW 2.
- Just like with HW 2 and HW 3, much of what you do on the midterm will end up in the final report, so spend the time to do a good job.

Simulation

Why Simulation?

- Up until now, when we do linear models, we used t-statistics, p-values, CIs
- These things are based on the sampling distribution of the estimators ($\hat{\beta}$) if the model is true and we don't do any model selection.
- What if we do model selection, use Kernels, think the model is wrong?
- None of those formulas work. And analogous formulas can be **impossible** (or painfully annoying) to derive.

Some simulation basics

```
set.seed(2018-02-20)
sample(1:10, replace=TRUE, prob=1:10/10)
```

```
##  [1]  6  8 10  9  7  6  3  9 10  8
```

```
sample(letters[1:10], replace=TRUE, prob=1:10/10)
```

```
##  [1] "i" "i" "a" "h" "e" "e" "h" "f" "d" "j"
```

```
sample(letters[1:10], replace=TRUE)
```

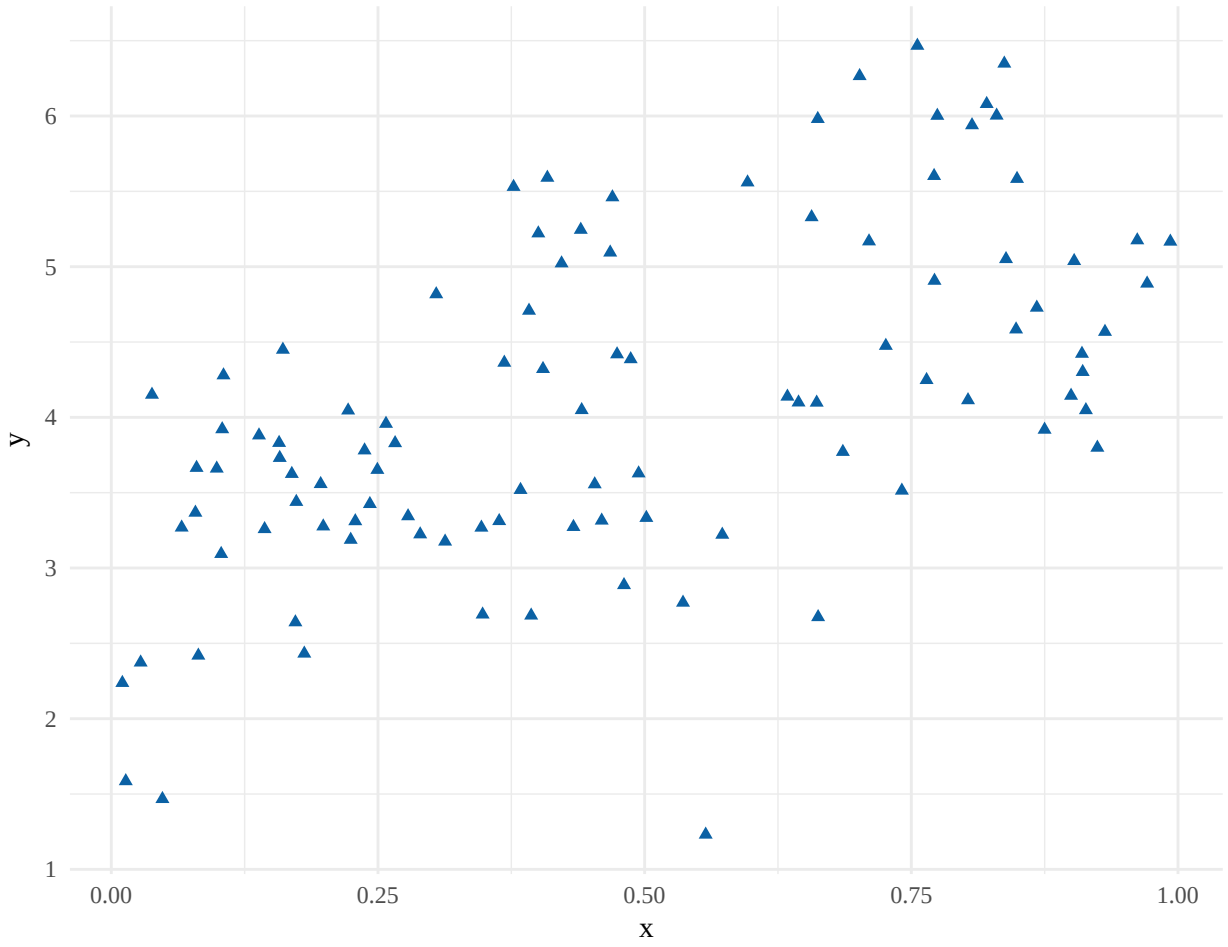
```
##  [1] "c" "e" "h" "j" "i" "b" "e" "a" "a" "j"
```

```
sample(letters[1:10])
```

```
##  [1] "g" "i" "b" "d" "c" "h" "e" "j" "a" "f"
```

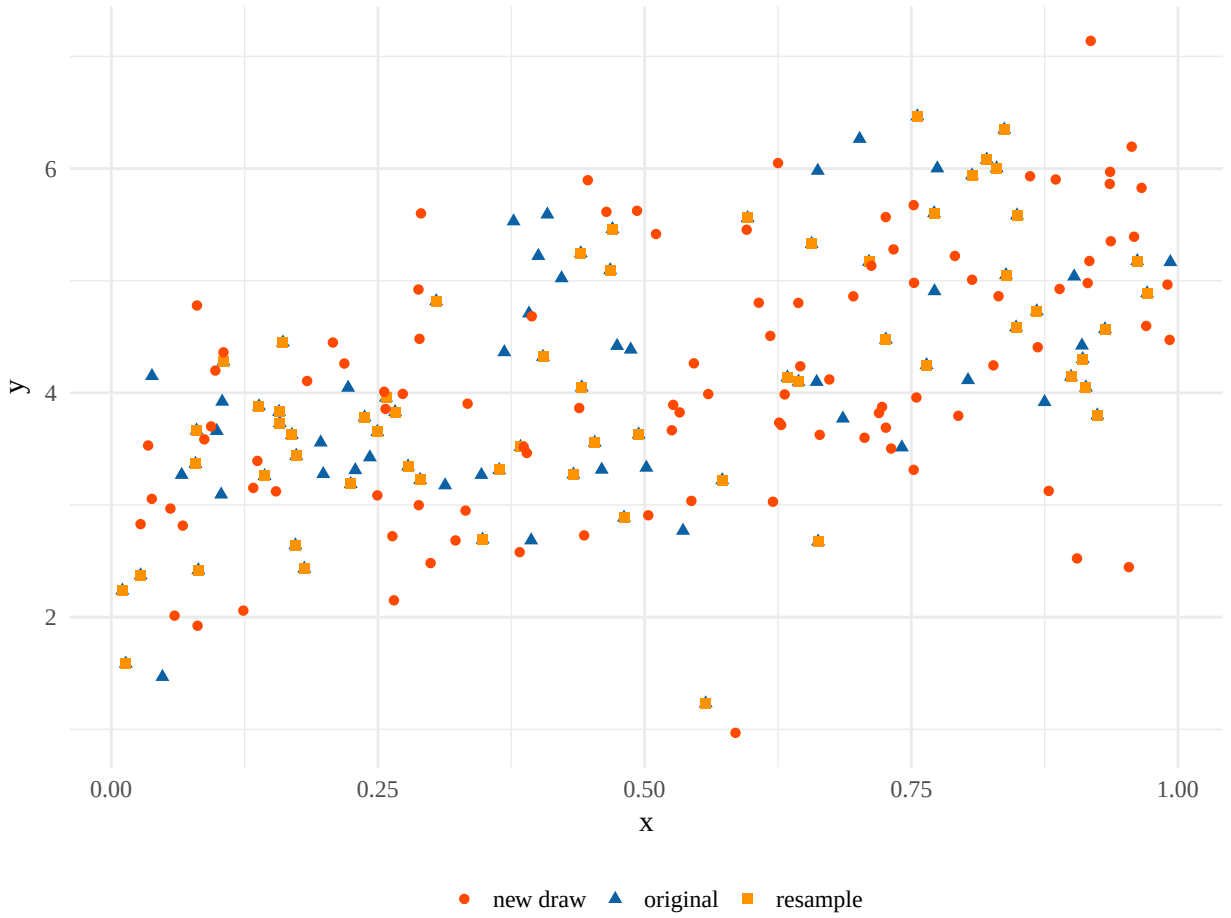
Resampling data

```
set.seed(2018-02-20)
n = 100; x = runif(n)
df = data.frame(x=x, y=3+2*x+rnorm(n))
ggplot(df, aes(x,y)) + geom_point(color=blue,shape=17)
```



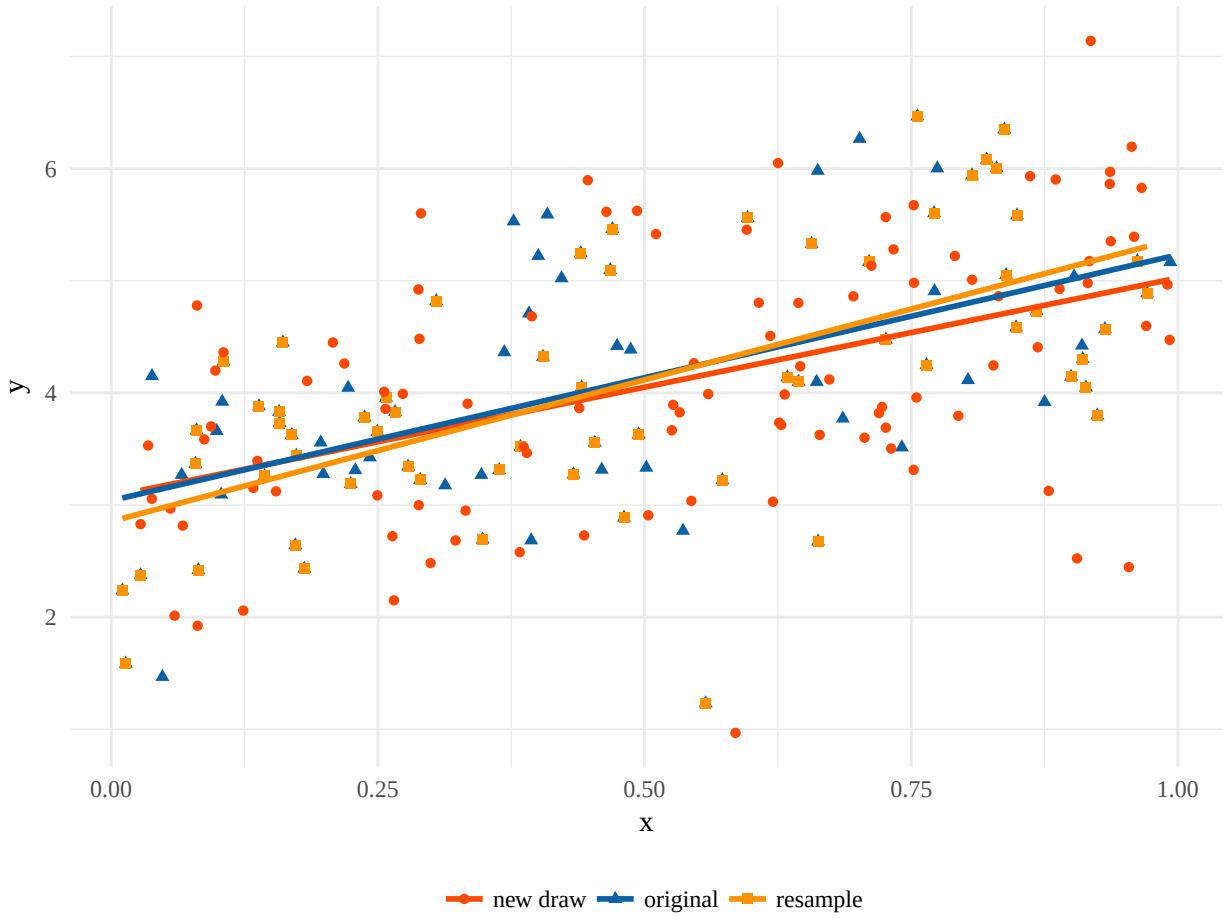
A sample (with replacement), and a new draw from the same distribution

```
resample <- function(df) {
  stopifnot(is.data.frame(df) | is.matrix(df))
  df[sample(1:nrow(df), replace = TRUE),]
}
df2 = resample(df)
xn = runif(n)
df3 = data.frame(x=xn, y=3+2*xn+rnorm(n))
df = rbind(df,df2,df3)
df$grp = rep(c('original','resample','new draw'), each=n)
p <- ggplot(df, aes(x,y,color=grp)) + geom_point(aes(shape=grp)) +
  scale_color_manual(values=c(red,blue,orange)) +
  theme(legend.title = element_blank(),legend.position = 'bottom')
p
```



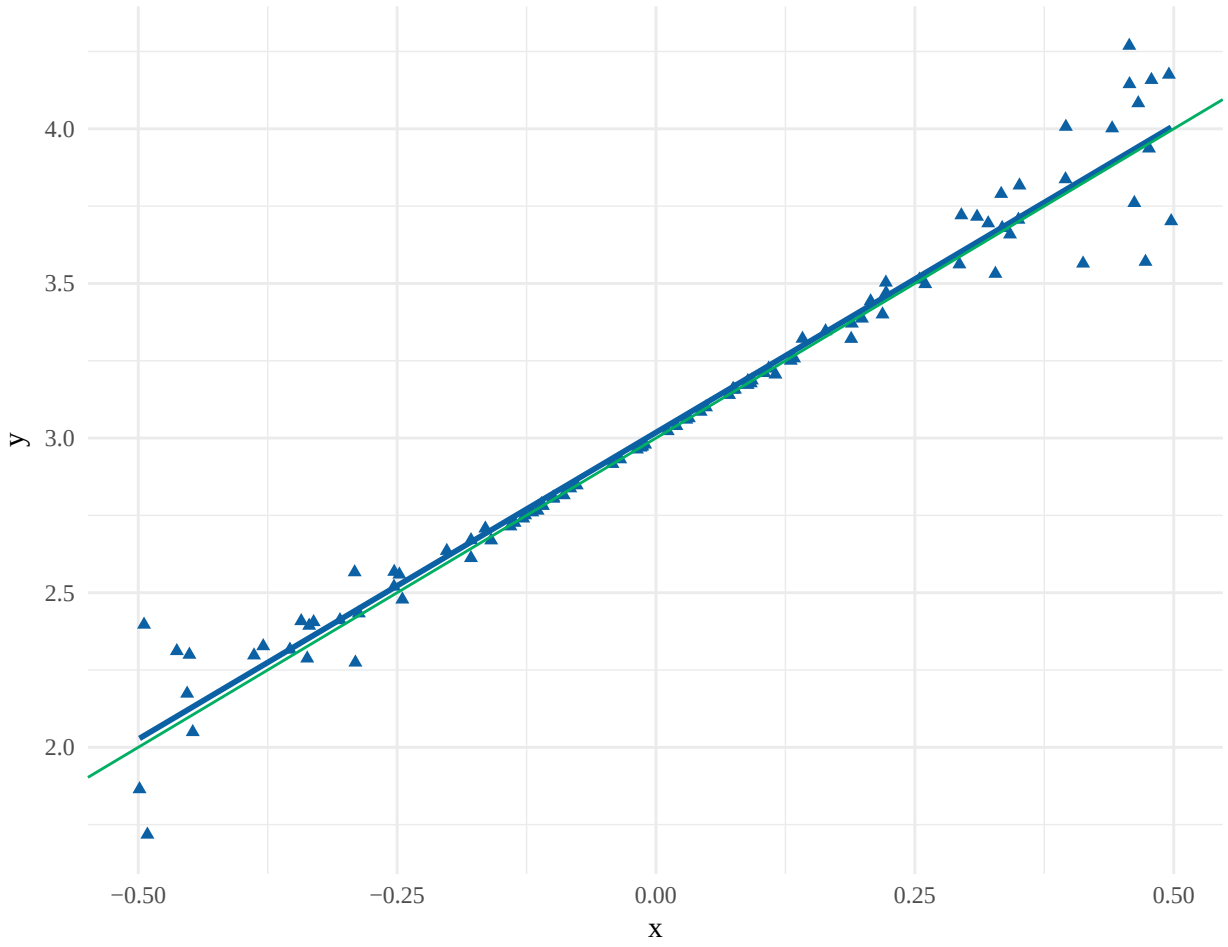
Add some lines

```
p + geom_smooth(method='lm', se = FALSE)
```



Using simulations to check modeling assumptions

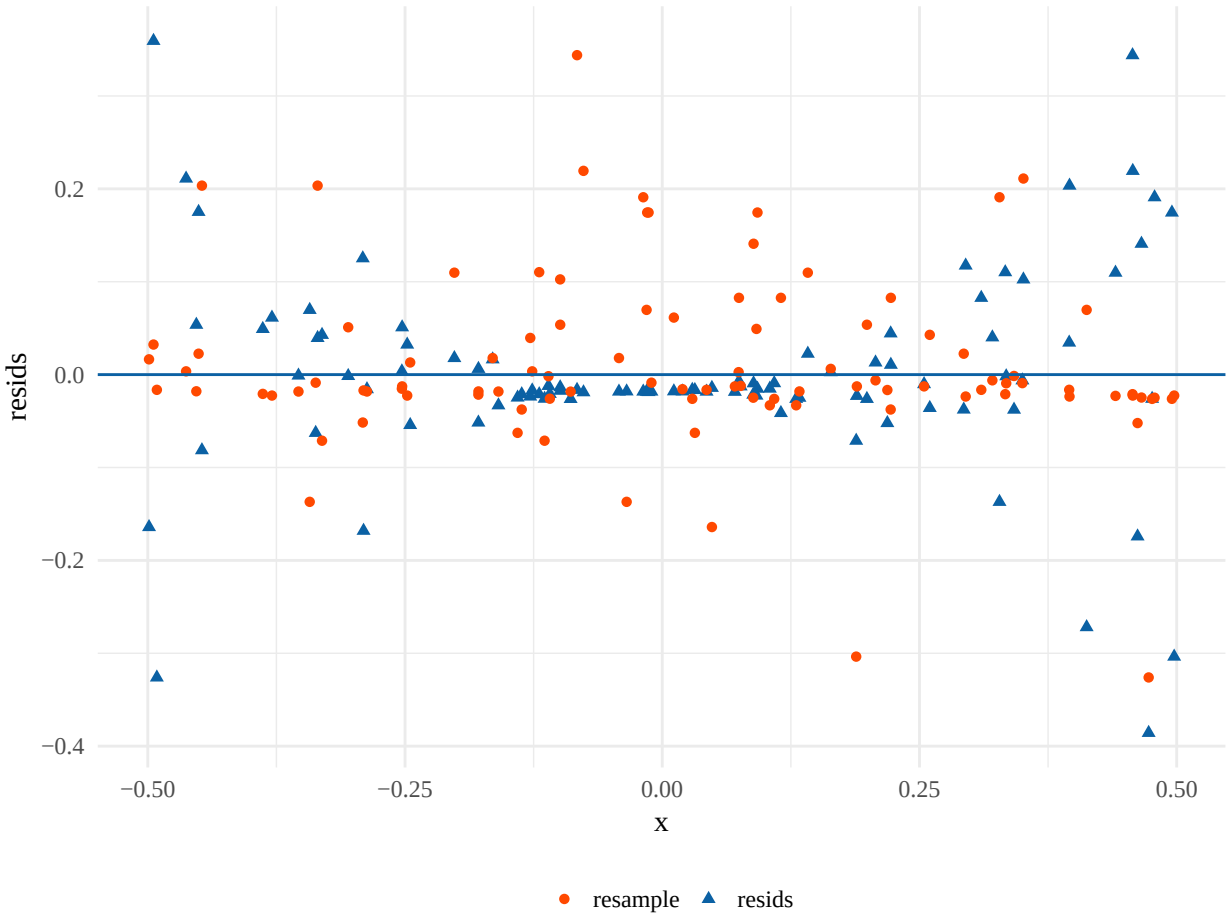
```
x = runif(n) - 0.5; y = 3+2*x + rnorm(n)*x^2
dfHetero = data.frame(x=x, y=y)
ggplot(dfHetero, aes(x,y)) + geom_point(color=blue,shape=17) +
  geom_smooth(method='lm', se=FALSE,color=blue) +
  geom_abline(intercept = 3, slope=2, color=green)
```



If the noise is homoskedastic...

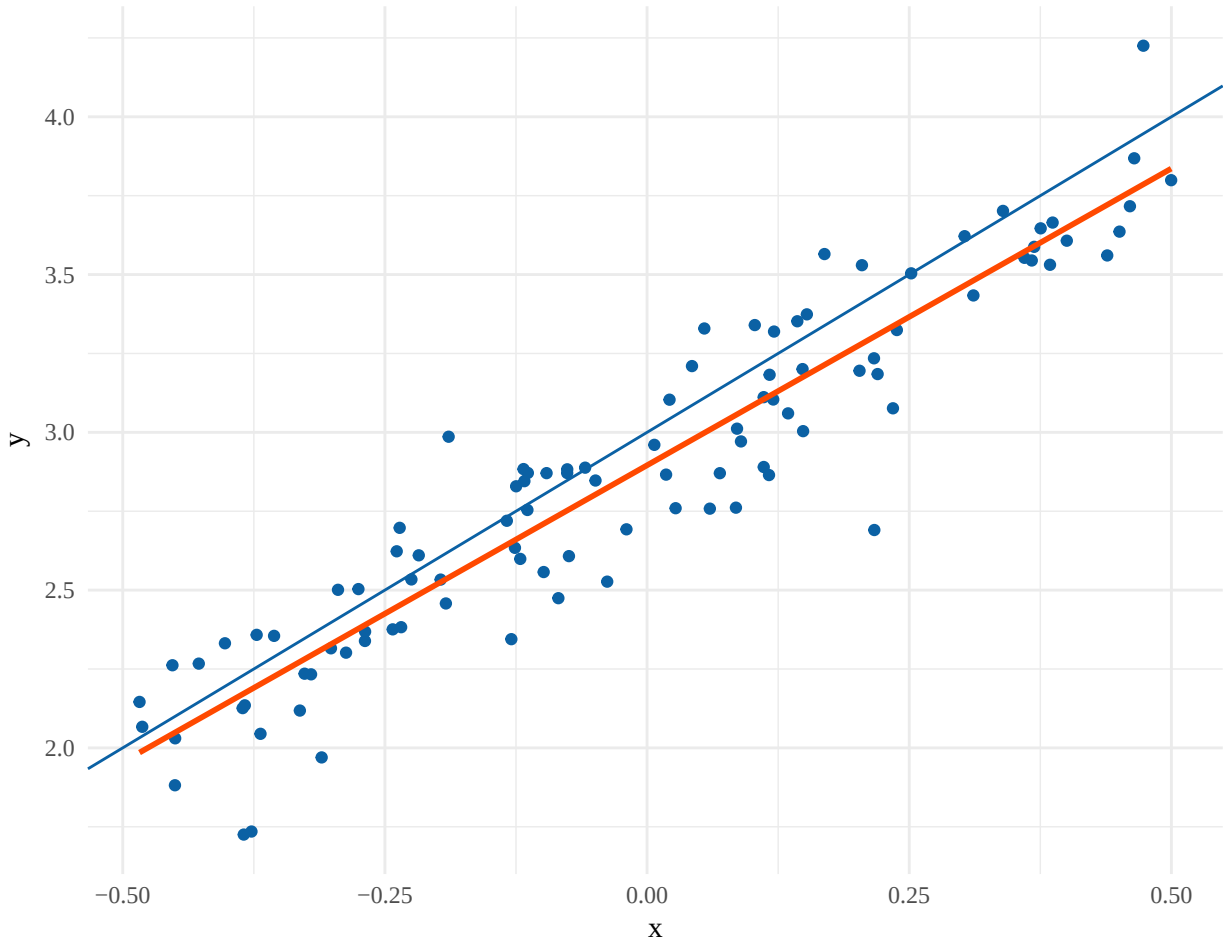
- The red and blue points should have the same distribution

```
heteromod = lm(y~x,data=dfHetero)
dfHetero$resids = residuals(heteromod)
dfHetero$resample = sample(residuals(heteromod), replace = TRUE)
dfHetero %>% gather(key='type', value='resids',-c(y,x)) %>%
  ggplot(aes(x,resids,color=type,shape=type)) + geom_point() +
  scale_color_manual(values=c(red,blue)) +
  theme(legend.title = element_blank(),legend.position = 'bottom') +
  geom_hline(yintercept=0, color=blue)
```



That one was easy

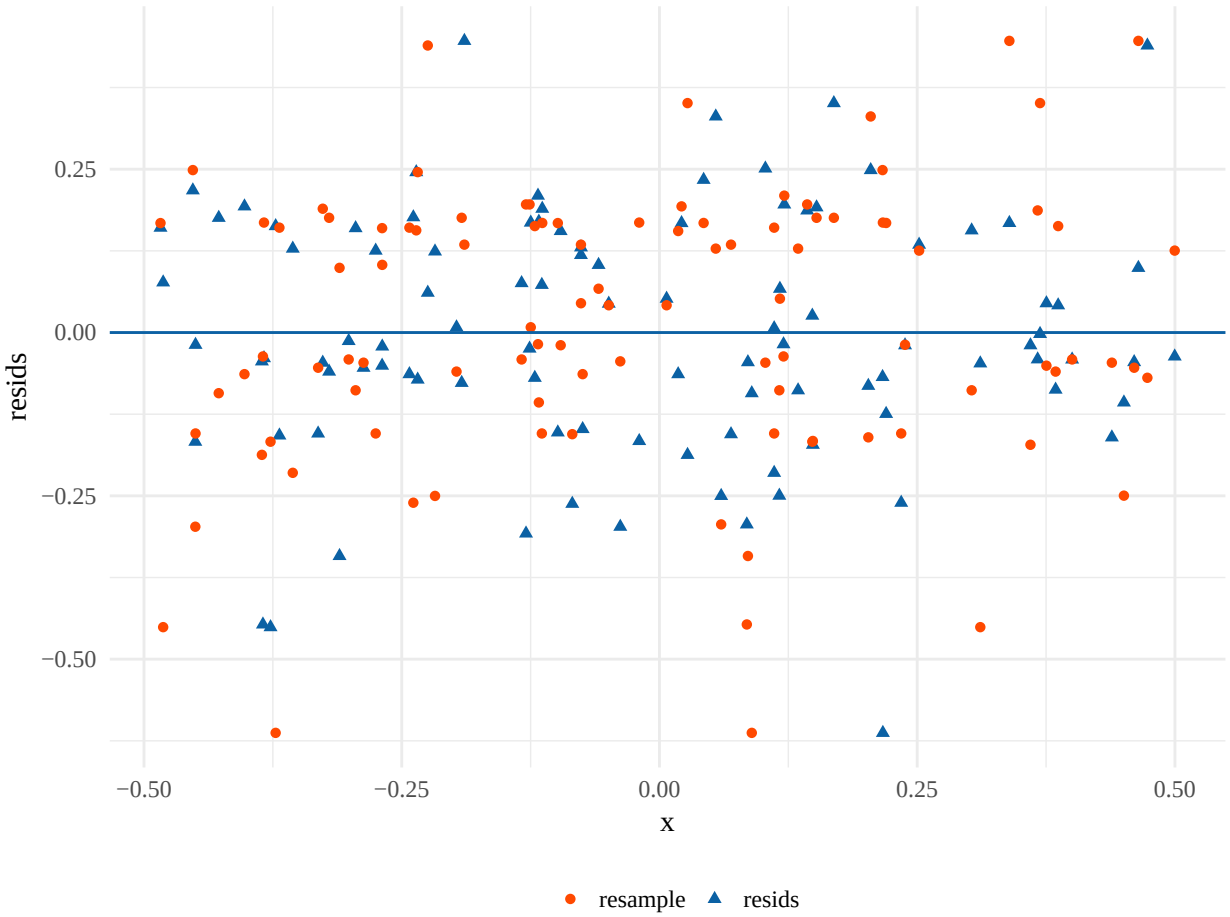
```
x = runif(n)-0.5
y = 3+2*x + c(arima.sim(list(ar=.8), n, rand.gen = function(n) 0.1* rt(n, df=5)))
dfTS = data.frame(x=x, y=y)
ggplot(dfTS, aes(x,y)) + geom_point(color=blue) +
  geom_smooth(method='lm',se=FALSE, color=red) +
  geom_abline(intercept = 3, slope = 2, color=blue)
```



If the noise is homoskedastic...

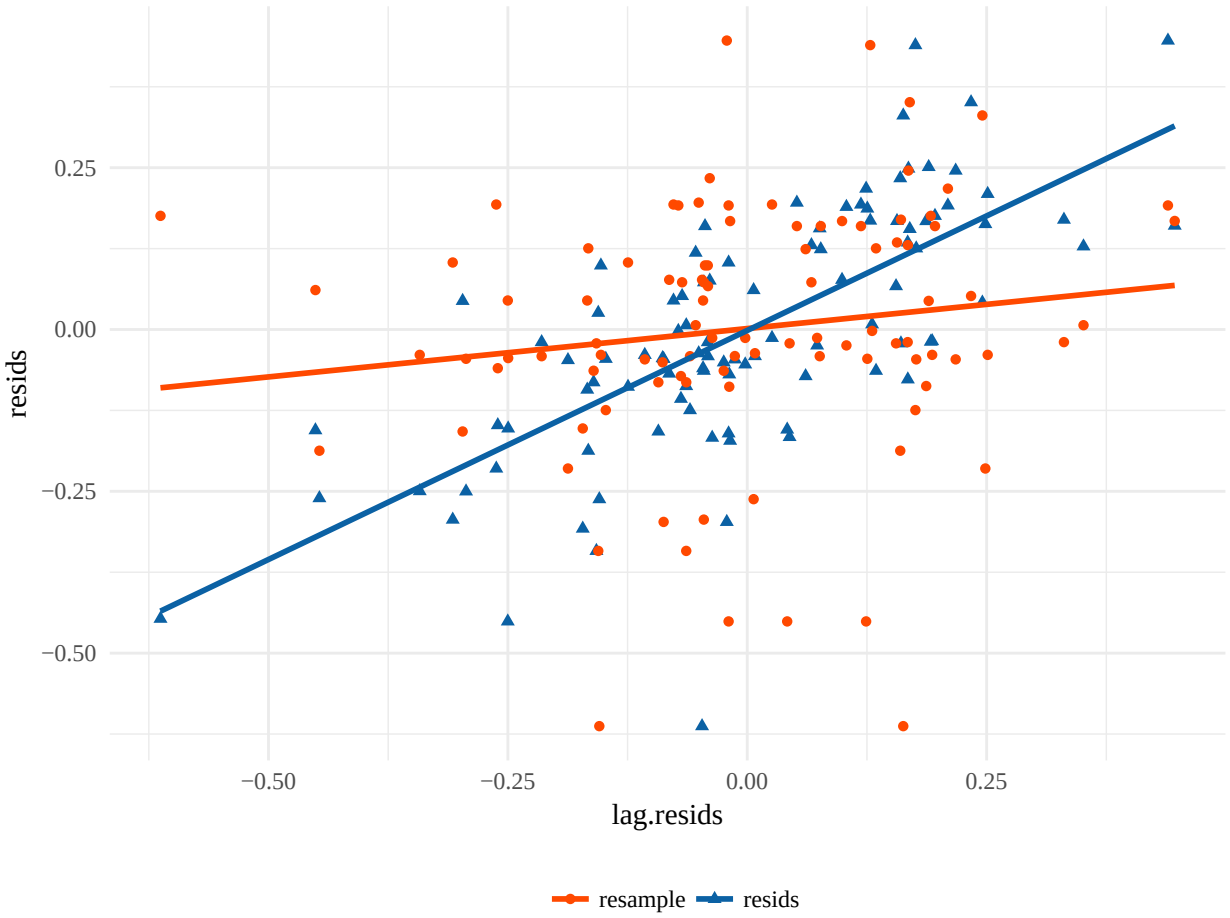
- The red and blue points should have the same distribution

```
tsMod = lm(y~x, data=dfTS)
dfTS$resids = residuals(tsMod)
dfTS$resample = sample(residuals(tsMod), replace = TRUE)
dfTS %>% gather(key='type', value='resids',-c(y,x)) %>%
  ggplot(aes(x,resids,color=type,shape=type)) + geom_point() +
  scale_color_manual(values=c(red,blue)) +
  theme(legend.title = element_blank(),legend.position = 'bottom') +
  geom_hline(yintercept=0, color=blue)
```



But...

```
lag.resids = with(dfTS, data.frame(lag.resids = resids[-n], resids = resids[-1]))
lag.resids$resample = sample(lag.resids$resids, replace = TRUE)
lag.resids %>% gather(key='type', value='resids',-lag.resids) %>%
  ggplot(aes(lag.resids,resids,color=type,shape=type)) + geom_point() +
  scale_color_manual(values=c(red,blue)) +
  theme(legend.title = element_blank(),legend.position = 'bottom') +
  geom_smooth(method='lm',se=FALSE)
```



Another useful command

```
sample.int(10)
```

```
##  [1] 10  5  8  1  4  9  7  3  2  6
```

What's the deal with this Bootstrap?

What's the deal with this Bootstrap?

- Suppose I want to estimate something and get a CI.
- But I don't know how to calculate the CI (or maybe I do, but it's hard)
- Then what?

Example 1

- Let $X_i \sim \chi^2_4$.
- I know if I estimate the mean with $\bar{X}$, then by the CLT (if n is big),

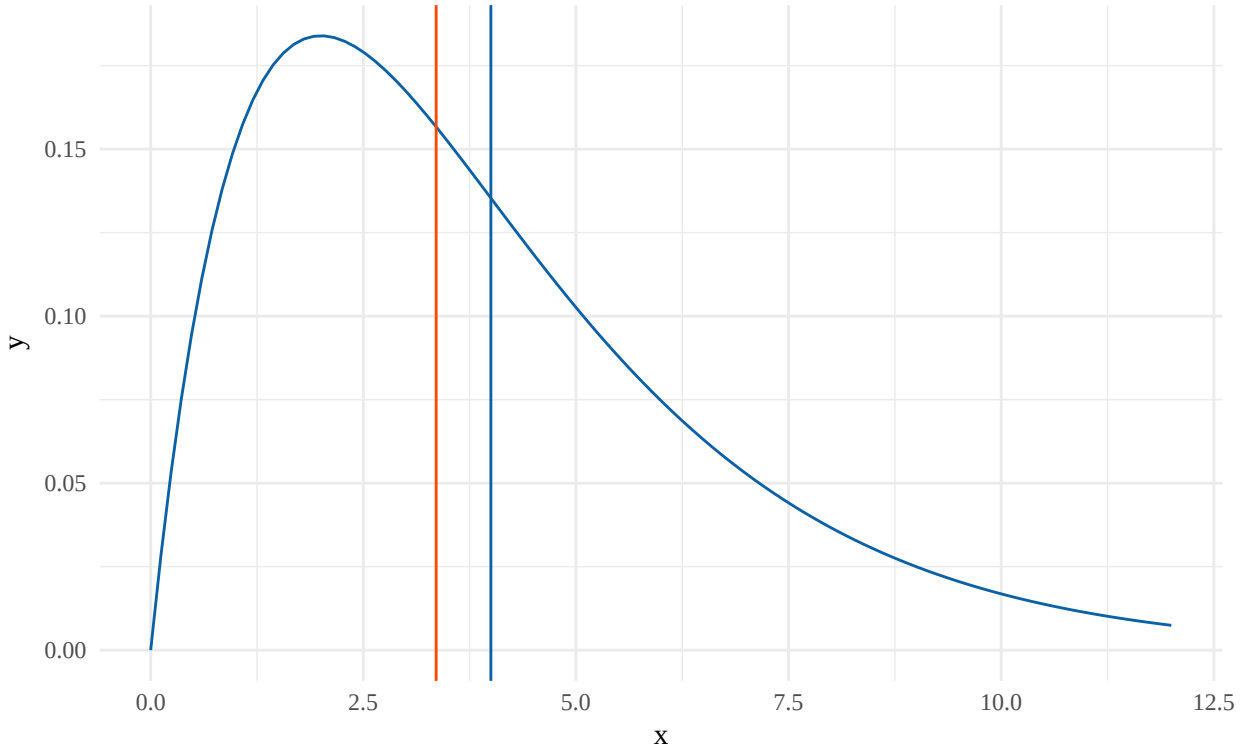
$$\frac{\sqrt{n}(\bar{X} - \mathbb{E}[X])}{s} \approx N(0,1).$$

- This gives me a 95% confidence interval like

$$\bar{X} \pm 2 * s/\sqrt{n}$$

- But I don't want to estimate the mean, I want to estimate the median.

```
ggplot(data.frame(x=c(0,12)), aes(x)) +
  stat_function(fun=function(x) dchisq(x, df=4), color=blue) +
  geom_vline(xintercept = 4, color=blue) + # mean
  geom_vline(xintercept = qchisq(.5,4), color=red) # median
```



Now what

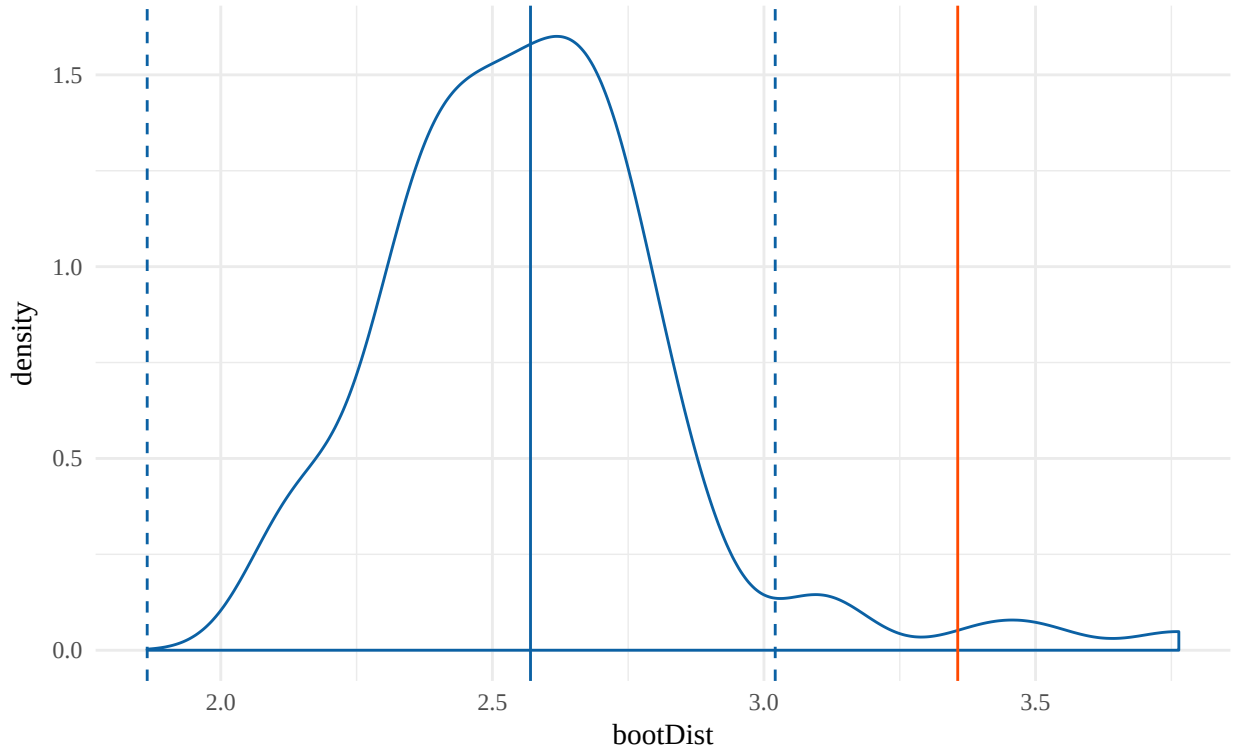
- I give you a sample of size 50, you give me the sample median.
- How do you get a CI?
- You can use the bootstrap!

```
set.seed(2018-02-20)
x = rchisq(n, 4)
(med = median(x))
```

```
## [1] 2.570231
```

```
B = 100
alpha = 0.05
bootMed <- function(x) median(sample(x, replace=TRUE))
bootDist = replicate(B, bootMed(x))
bootCI = 2* med - quantile(bootDist, probs = c(1-alpha/2, alpha/2))
ggplot(data.frame(bootDist), aes(bootDist)) + geom_density(color=blue) +
  geom_vline(xintercept = bootCI, col=blue, linetype=2) +
```

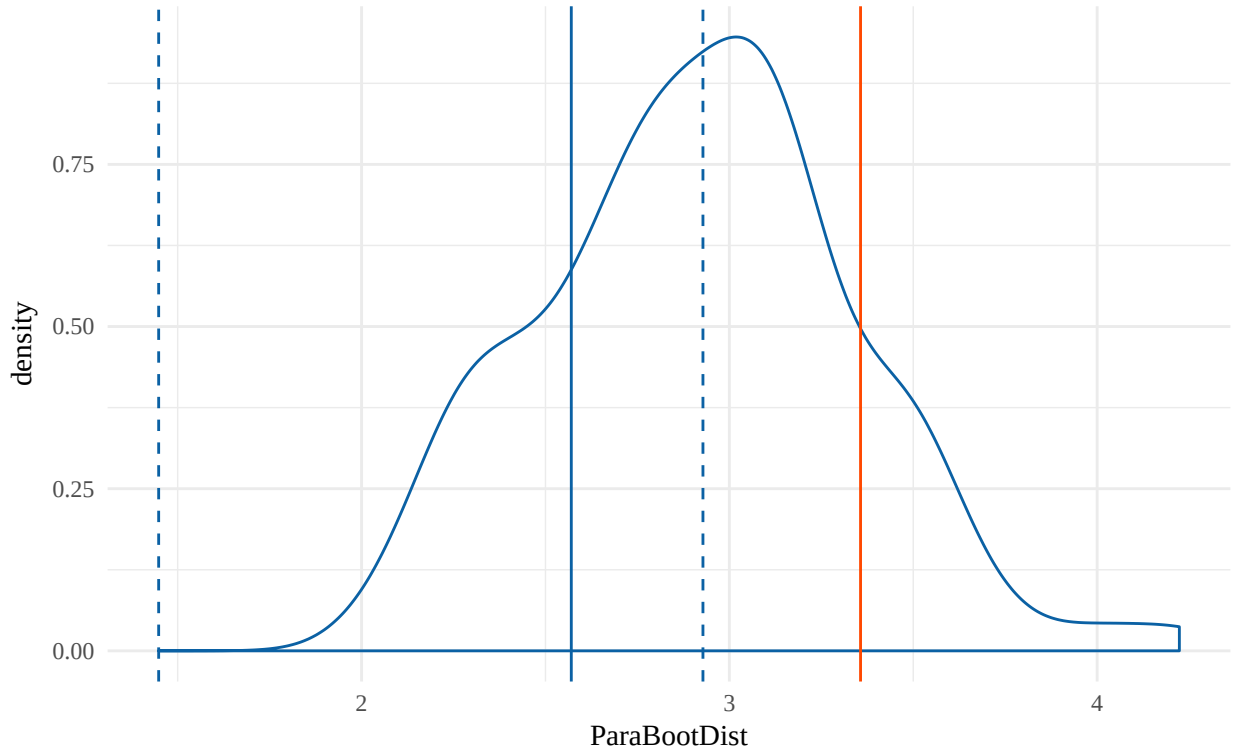
```
  geom_vline(xintercept = med, col=blue) +
  geom_vline(xintercept = qchisq(.5, 4), col=red) # truth
```



An alternative

- In that bootstrap, I didn't use any information about the data-generating process.
- What if I told you that the data came from a χ^2, but I didn't tell you the degrees of freedom?
- You could try a "parametric" bootstrap:

```
xbar = mean(x)
s = sd(x)
ParaBootSamp <- function(B, xbar, s){
  means = rnorm(B, mean=xbar, sd=s/sqrt(n))
  meds = qchisq(.5, means)
  return(meds)
}
ParaBootDist = ParaBootSamp(B, xbar, s)
ParaBootCI = 2* med - quantile(ParaBootDist, probs = c(1-alpha/2, alpha/2))
ggplot(data.frame(bootDist), aes(ParaBootDist)) + geom_density(color=blue) +
  geom_vline(xintercept = ParaBootCI, col=blue, linetype=2) +
  geom_vline(xintercept = med, col=blue) +
  geom_vline(xintercept = qchisq(.5, 4), col=red) # truth
```



In truth

- Let's compare these intervals
- The nonparametric bootstrap (first one) had a width of

```
bootCI[2] - bootCI[1]
```

```
##     2.5%
## 1.156143
```

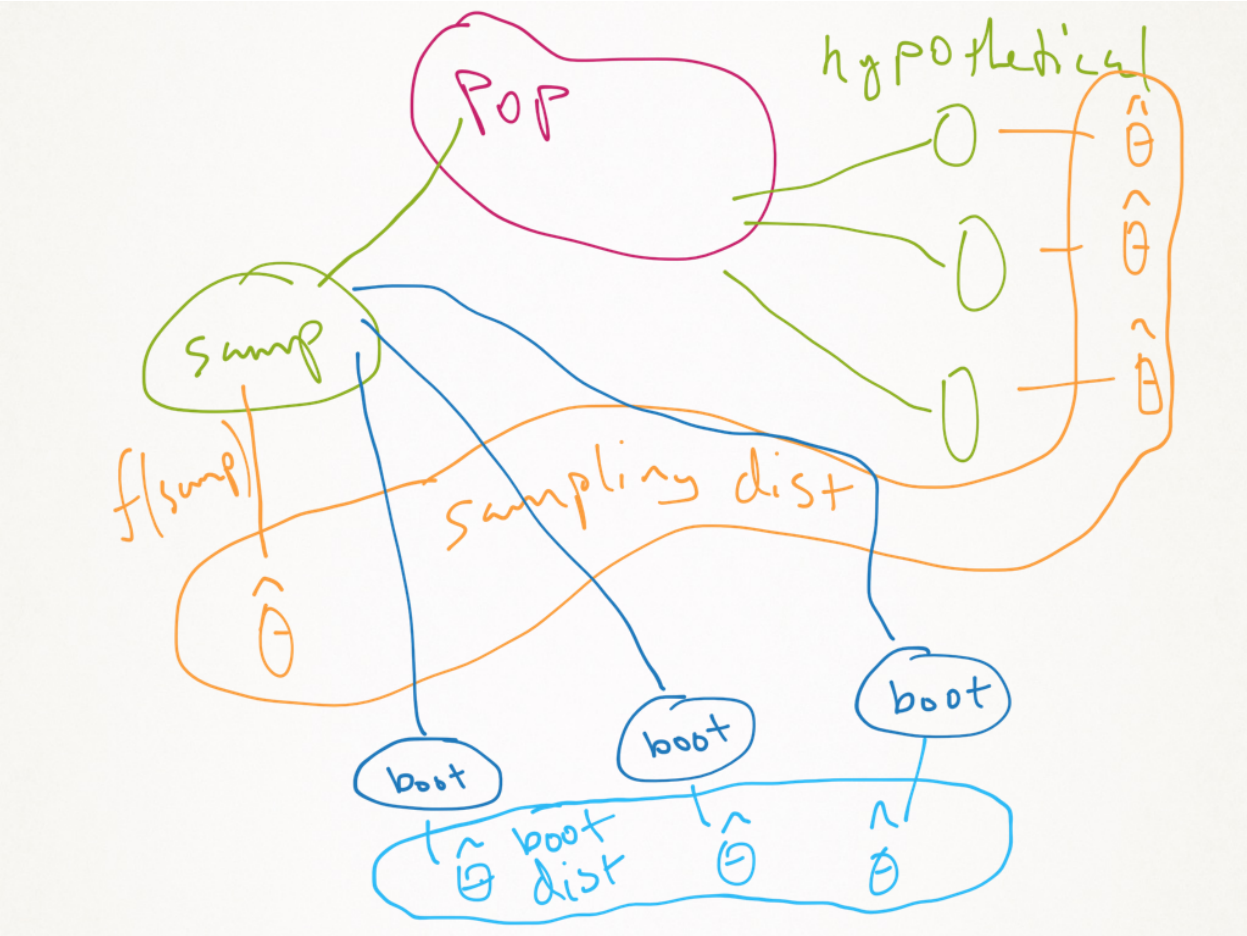
- The parametric bootstrap (second one) had a width of

```
ParaBootCI[2] - ParaBootCI[1]
```

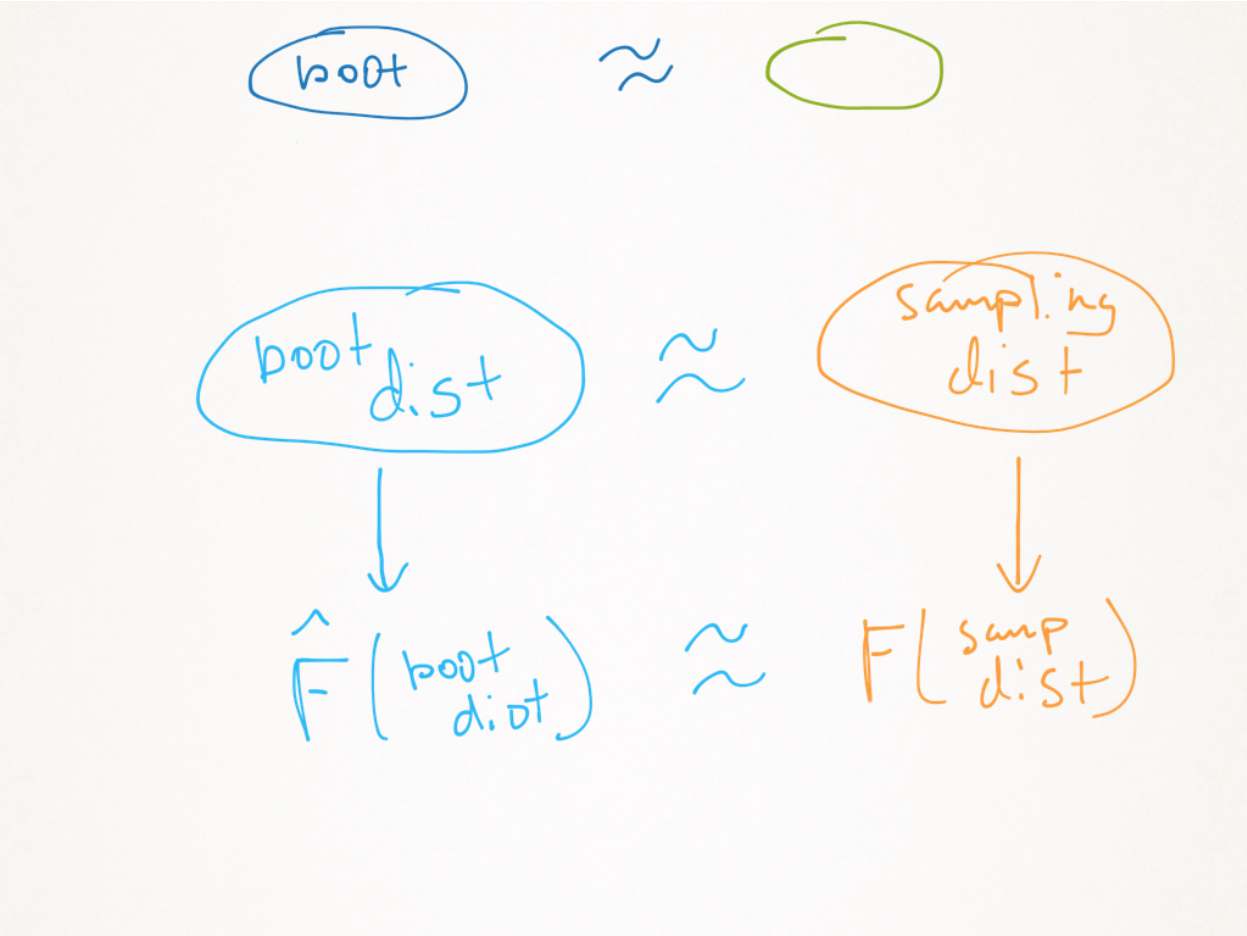
```
##     2.5%
## 1.479937
```

- Using theory, we could find the exact CI. In this case, it has a width of 1.76.

Bootstrap diagram



Bootstrap intuition



Bootstrap error sources

(From the bottom up on the last slide)

1. Simulation error: using only B samples to estimate F with $\hat{F}$.
2. Statistical error: our data depended on a sample from the population. We don't have the whole population so we make an error by using a sample (Note: this part is what **always** happens with data, and what the science of statistics analyzes.)
3. Specification error: If we use the model based bootstrap, and our model is wrong, then we think we are badly overconfident in our assessment of error.

Recap

- There are essentially 2 types of bootstrap
  1. Parametric
  2. Nonparametric
- If you **really** believe your model, use the first
- If not, use the second

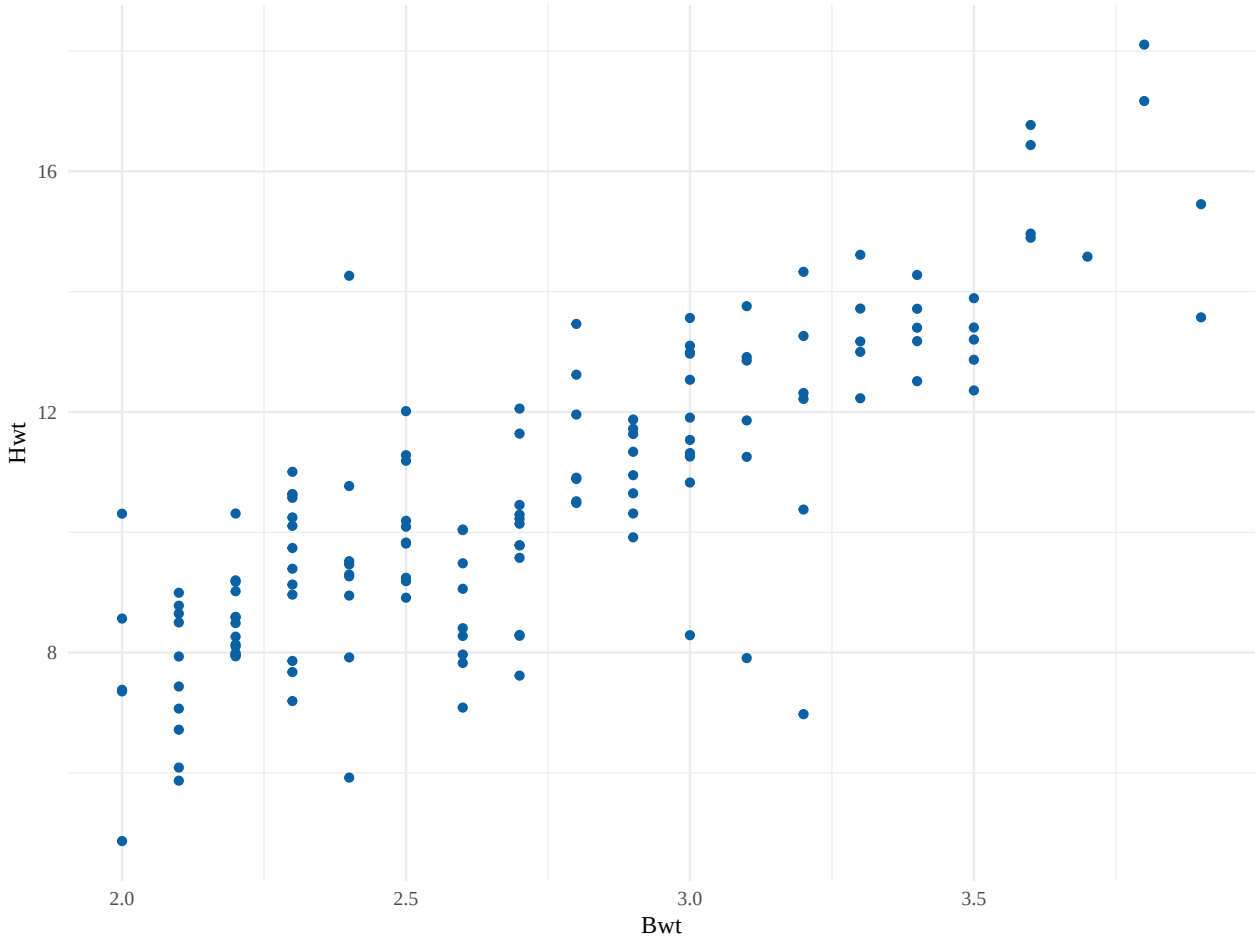
- Both are valid

Example 2

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(MASS)
ggplot(fatcats, aes(Bwt, Hwt)) + geom_point(color=blue)
```



A model

```
cats.lm <- lm(Hwt ~ 0+Bwt,data=fatcats)
summary(cats.lm)
```

```
##
## Call:
## lm(formula = Hwt ~ 0 + Bwt, data = fatcats)
```

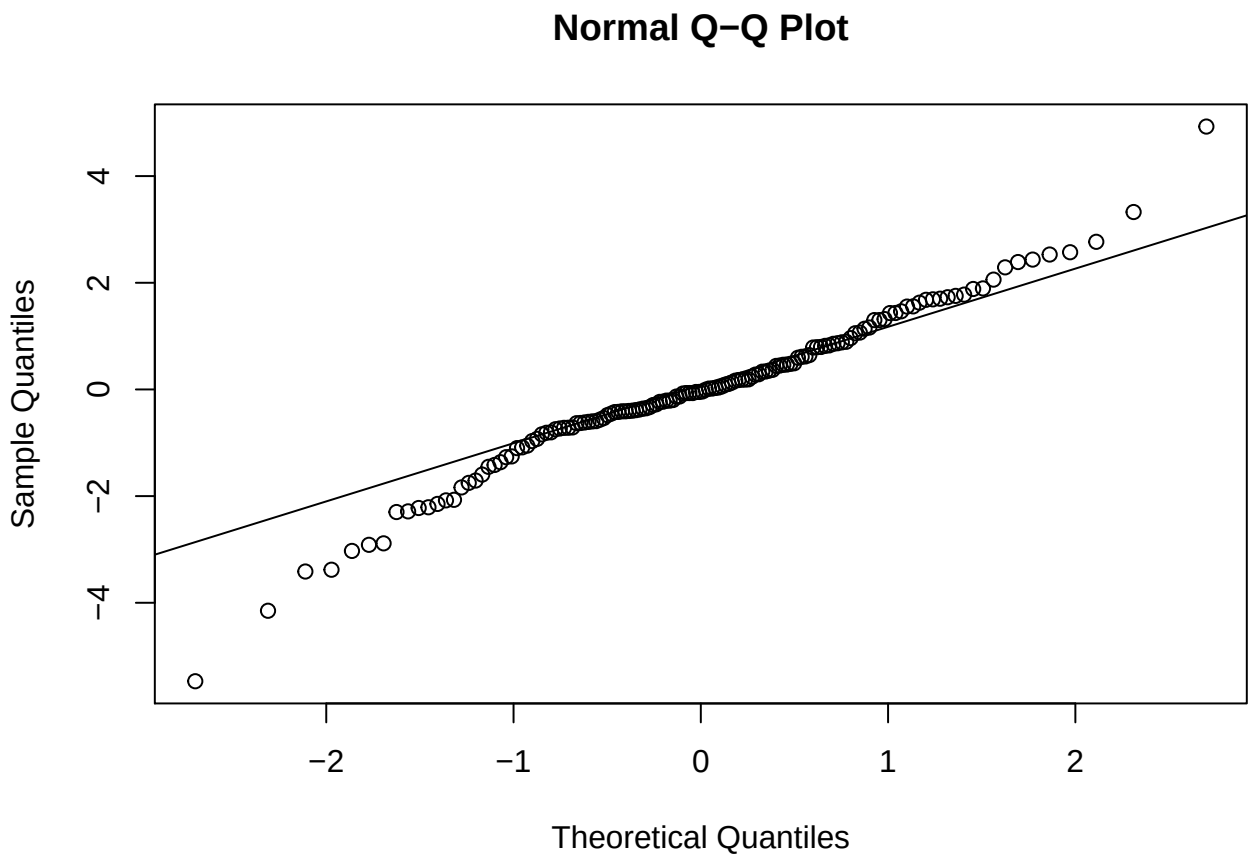
```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.4713 -0.6530 -0.0427  0.8189  4.9289
##
## Coefficients:
##     Estimate Std. Error t value Pr(>|t|)
## Bwt  3.88959    0.04385    88.7   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.456 on 143 degrees of freedom
## Multiple R-squared:  0.9821, Adjusted R-squared:  0.982
## F-statistic:  7868 on 1 and 143 DF,  p-value: < 2.2e-16
```

```
confint(cats.lm)
```

```
##        2.5 %   97.5 %
## Bwt 3.802907 3.976266
```

I think that that CI is wrong...

```
qqnorm(residuals(cats.lm))
qqline(residuals(cats.lm))
```



Model-based bootstrap

```
catsResids <- function(){
  resids = residuals(cats.lm)
  newResids = sample(resids, replace=TRUE)
    # resample the residuals from the original model
  newCats = data.frame(Bwt = fatcats$Bwt,
    Hwt=fitted(cats.lm) + newResids) # create a new dataframe
                                     # with the original x's but new y's
  return(newCats)
}
fitCats <- function(newCats) coef(lm(Hwt~0+Bwt, data=newCats)) # get the coef from OLS
fitCats(fatcats) # test the above on original data, should give same coef
```

```
##      Bwt
## 3.889586
```

Model-based bootstrap

```
catsBoot <- function(B, alpha, func, bootname){
  coefs = replicate(B, fitCats(func())) # the bootstrap distribution hat(F)
  ci = 2*coef(cats.lm) - quantile(coefs, probs = c(1-alpha/2, alpha/2)) # ci
  cis = rbind(confint(cats.lm), ci) # The rest just for comparison
  rownames(cis) = c('original',bootname)
  return(cis)
}
cisPara = catsBoot(1000,.05, catsResids, 'MBB') # do it!
```

Nonparametric bootstrap

```
resampData <- function(){
  sampled.rows = sample(1:nrow(fatcats), replace=TRUE) # resample original data
  new.cats = fatcats[sampled.rows,]
}
cisNonPara = catsBoot(1000, .05,  resampData, 'NonParaB')
  # use the prev func to
  # bootstrap on resampled data
cisPara
```

```
##             2.5 %   97.5 %
## original 3.802907 3.976266
## MBB      3.808350 3.982101
```

```
cisNonPara
```

```
##             2.5 %   97.5 %
## original 3.802907 3.976266
## NonParaB 3.798535 3.975114
```

Bootstrapping with nonparametric regression

- This is a bit harder
- The reason is that we use CV to choose the bandwidth
- So we have to repeat that step in the bootstrapping
- That is:
    1. Input data
    2. Use CV to choose a smoothing parameter
    3. Use the chosen parameter to estimate the smooth function
    4. Resample the data
    5. Using this new data, repeat 2 and 3