201ab Quantitative methods

Resampling
Randomization
Bootstrap
Cross-validation

Why are we doing "statistics"?

- We can calculate a statistic on the data – that's an arbitrary function… why don't we just call it a day?
  - There is variability in our measurements: noisy error in measurements, random sampling of populations, stochastic processes in the world.
- Want to separate signal from noise, to figure out if our data has certain structure or if that structure is just noise
  - Null hypothesis testing, model selection, etc.
- Want to quantify the uncertainty / error / confidence in our statistics / estimates.
  - Parameter estimation, confidence intervals, etc.
- Want to predict future data, and predict our accuracy
  - Danger of fitting and predicting noise (overfitting).

Why are we doing "statistics"?

- How to quantify uncertainty and account for variability/noise/error in our measurements?
  - Classical frequentist stats: define models, derive their long-run frequency behavior using probability, to analytically obtain sampling distributions of statistic, under null, and of new data.
  - **Resampling methods: use existing data (perhaps with some invariances) as guess of population distribuion, and generate sampling distributions numerically.**
  - Bayesian methods: define models, and their probability of generating data, invert to calculate posterior probability of model/parameters given data, and posterior predictive distribution of new data.

Resampling

- Use the current data as an approximation of future data. "re"sample from current data in various ways to generate distributions.

- **Randomization**:
  - Build a null hypothesis distribution (and thus p-values) by shuffling the current data.
- **Bootstrap**:
  - Build sampling distribution of statistic by hallucinating alternate samples of data, get confidence interval that way.
- **Cross-validation**:
  - Simulate the process of checking a model on yet-unseen data by using one part to fit, another to validate.

Resampling: Logic

- How statistics works:
  - We get some data
  - Make assumptions about the data-generating process
  - Use the data to estimate properties of this process.
  - Use our assumptions and estimates to infer general properties of the process, and predict new outcomes.
- E.g.
  - We measure heights
  - We assume heights are normally distributed
  - We estimate the mean, s.d. of heights
  - We use the mean, s.d. via a normal distribution to make inferences and predictions.

Resampling: Logic

- How statistics works:
  - We get some data
  - Make assumptions about the data-generating process
  - Use the data to estimate properties of this process.
  - Use our assumptions and estimates to infer general properties of the process, and predict new outcomes.

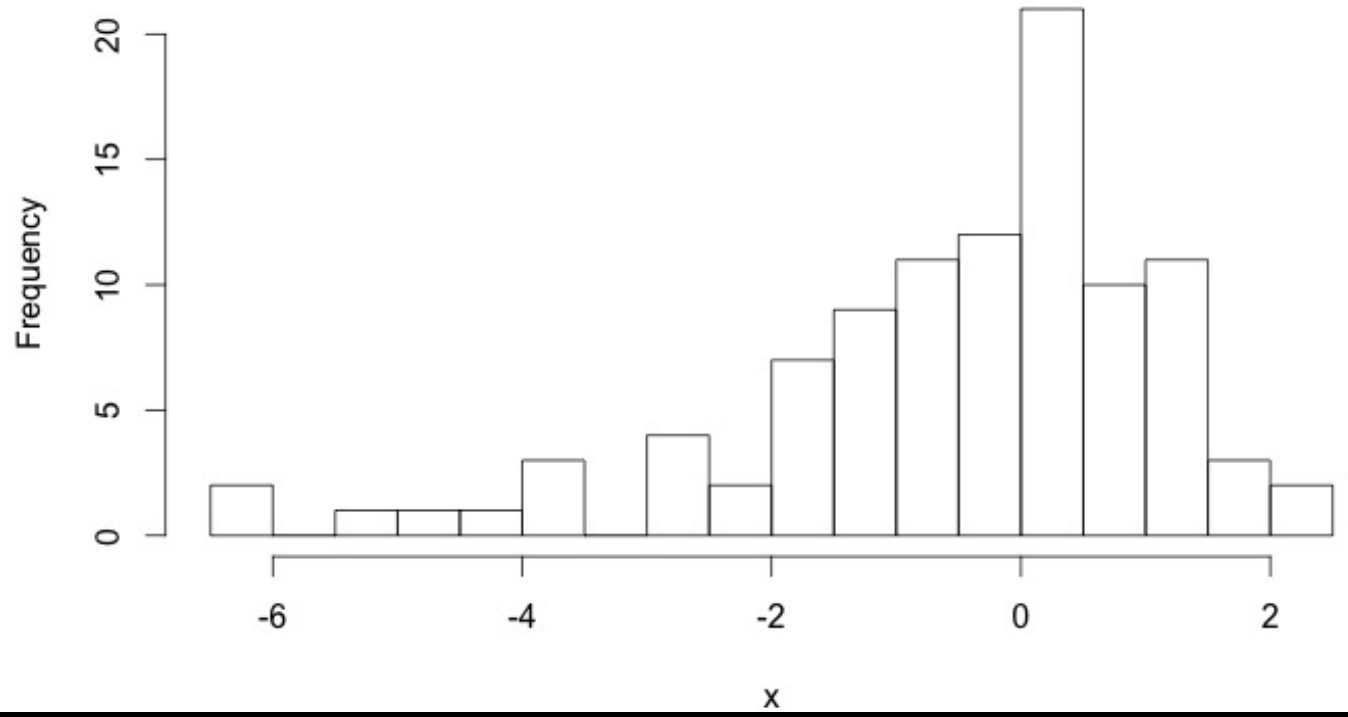
- How strong do our assumptions need to be?
  - Parametric assumptions: Data follow a *normal distribution?*
  - Let's just assume that future data will be like previous data.

Resampling: Logic

- Future data will be like the current data

```
x
```

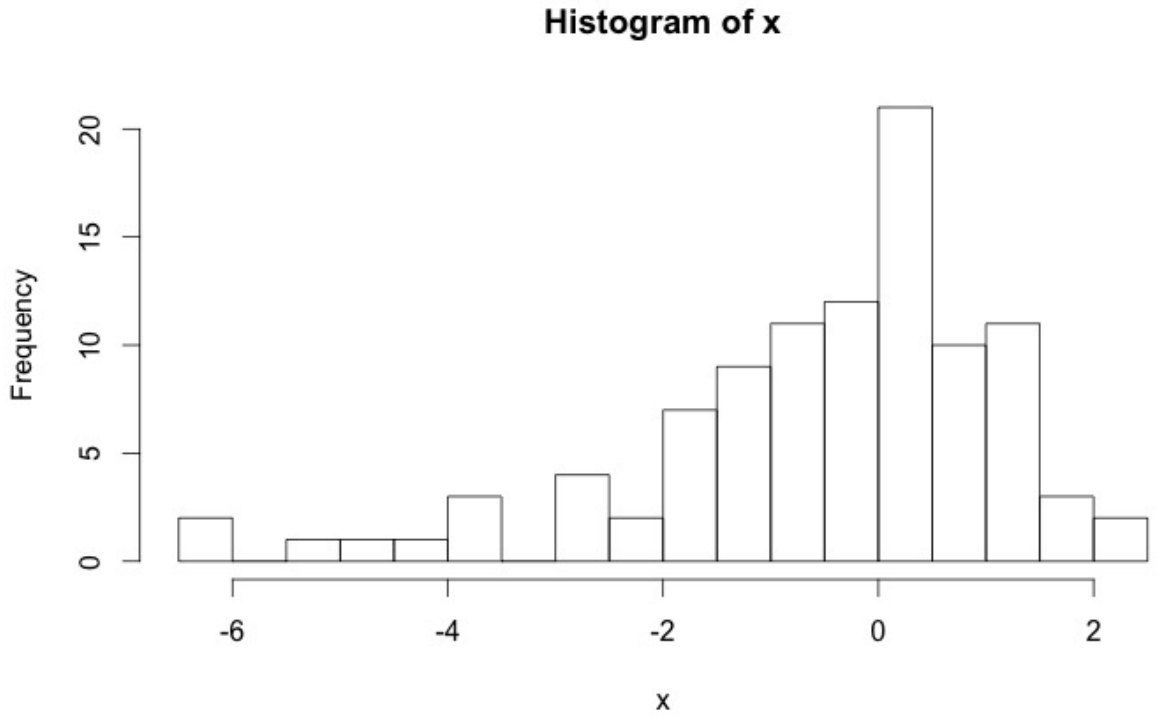
```
[100] -0.81057568 -3.75588056 -0.77649265 -2.71176279 -0.72528754 -1.79951633  0.28982537 -1.86105795 -0.08463999  0.08801584  0.42490192 -0.32787760  1.40340032
0.35678602 -1.41562830 -1.04574622  0.09898905  1.05035238 -2.16351814 -2.54987788  0.60558522  0.98092604 -0.61329110 -0.42499095 -0.75423395 -0.40166514
0.18090021 -0.57891140 -0.63355362  1.42385112  0.19384376 -1.50328405  1.16187994 -2.61733233  1.33129141 -0.63628871 -0.45647740 -5.07116647 -1.74510401 -
1.41756240  0.19368629 -0.09093373 -0.97658404 -6.49857275 -1.61333570  0.22196662 -0.24209335  0.75823578 -0.31382485  1.22549832  2.26054057  0.18598882
1.78827723  1.21081248 -2.81913736  1.97343288 -1.28652368  1.54594865  0.93246801  0.23824970  0.35839401  0.57989193 -4.64778399 -1.25622966 -4.42018876
0.20398443 -1.23291749 -0.44188558  0.61962049  0.81900205 -1.47554994  0.03681568 -2.06985538 -0.26888997 -1.32263459 -1.31015657  0.49810722  0.08016335
0.73409104 -0.37473450 -3.96311430 -3.79768621  0.06302523  1.16587685  1.47153014  0.54874491 -0.62163567 -1.88914946 -0.69980288  2.42535899 -1.59746565 -
6.15922478  0.40459299  1.05146166  0.47986380  0.44199065  0.91806090 -0.49377271  1.20637696  0.03993493
```




```
hist(x)
```

Resampling: Logic

- Future data will be like the current data



- Histogram of x is our best estimate of the probability distribution of future samples of x.
- More formally: PDF of X is a mixture of our data points.

Resampling: Logic

- Future data will be like the current data
- We can generate more samples by resampling from the PDF made up of our data.

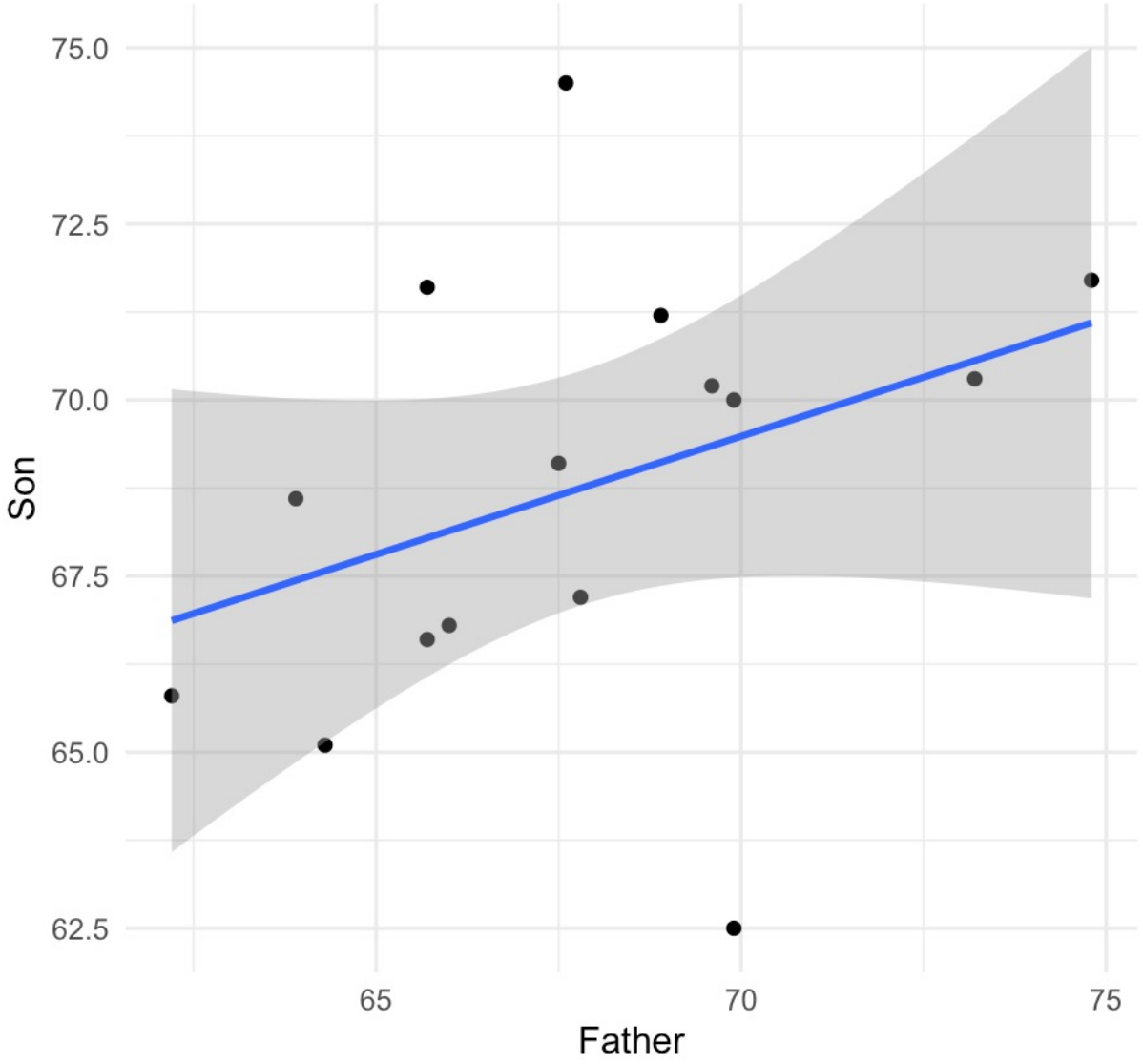
- By resampling in slightly different ways, we can get sampling distributions to…
  - …build null hypothesis distributions (randomization)
  - …get confidence intervals (bootstrapping)
  - …obtain prediction distributions (cross-validation)

- When we are building sampling distributions of some statistic, our resampled samples must be **of the same size as the real one, to obtain the sampling distribution of our statistic.**

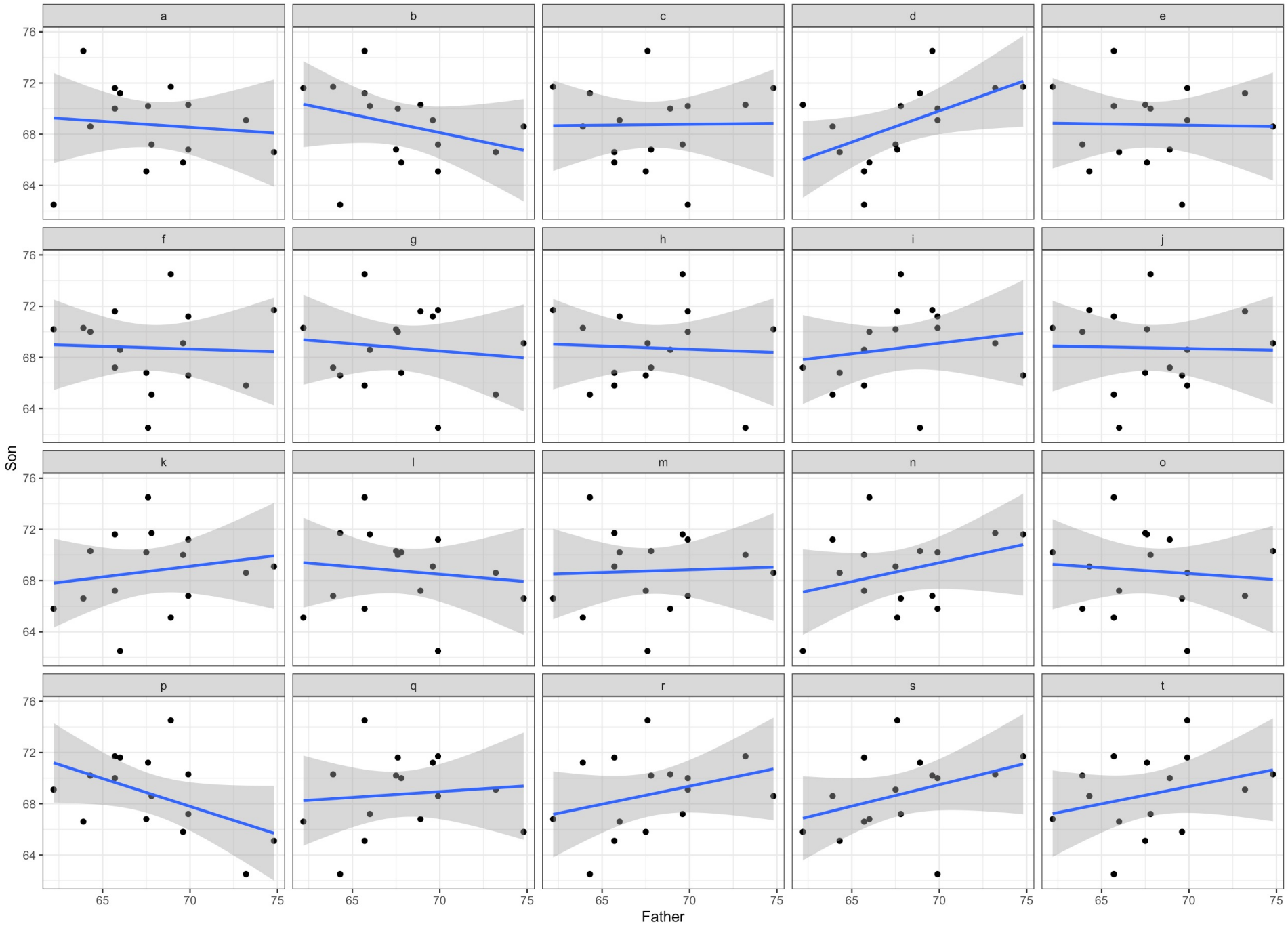
Resampling: Overview

- Randomization / Permutation / Shuffling:
  - Define a statistic that measures structure of interest
  - Resample so as to destroy the structure of interest
  - Calculate statistic on shuffled samples
  - Distribution of shuffled statistics is the null hypothesis sampling distribution of the statistic, compare statistic on real data to this
- Bootstrapping:
  - Draw more samples like the current one
  - Run some estimator on those samples
  - To build a sampling distribution of an estimate
  - This is useful for confidence intervals, etc.

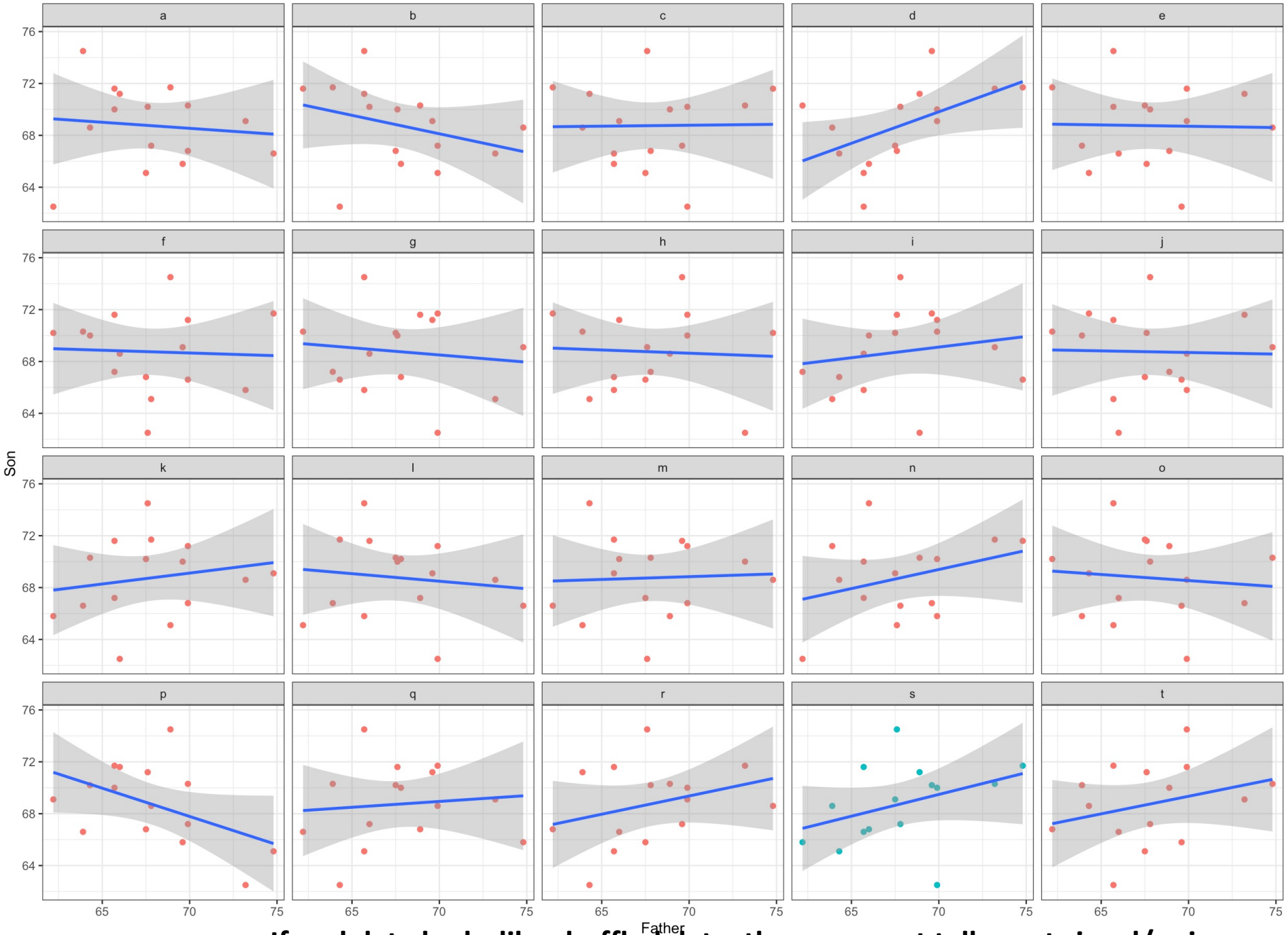
Randomization intuition.



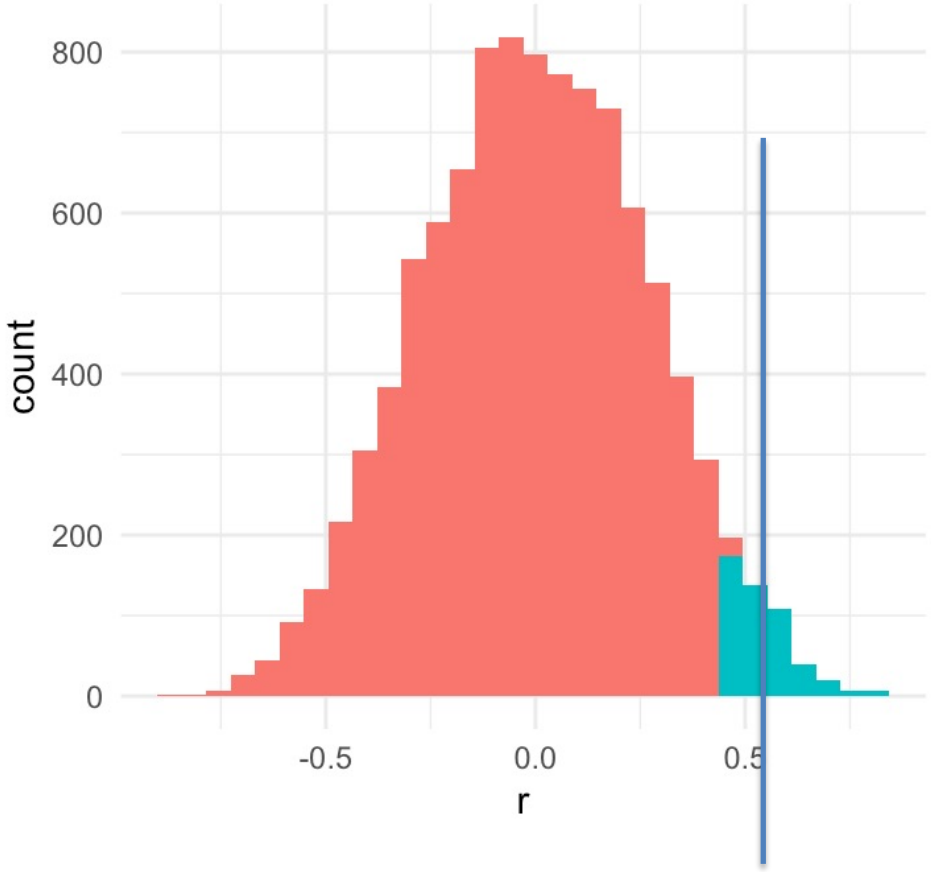
This is a scatterplot from the real data.



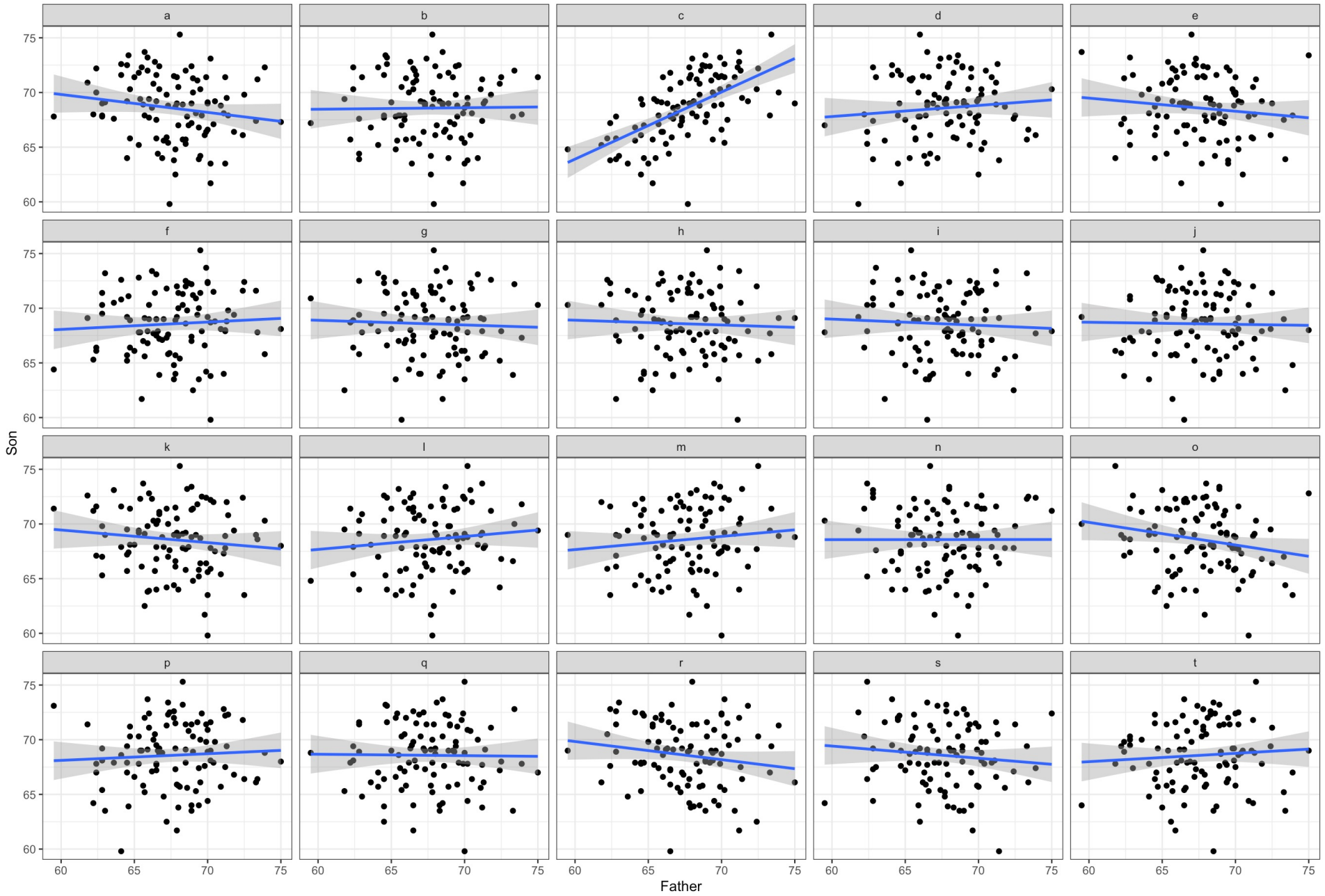
19 shuffled samples, and 1 real data. Which is real?



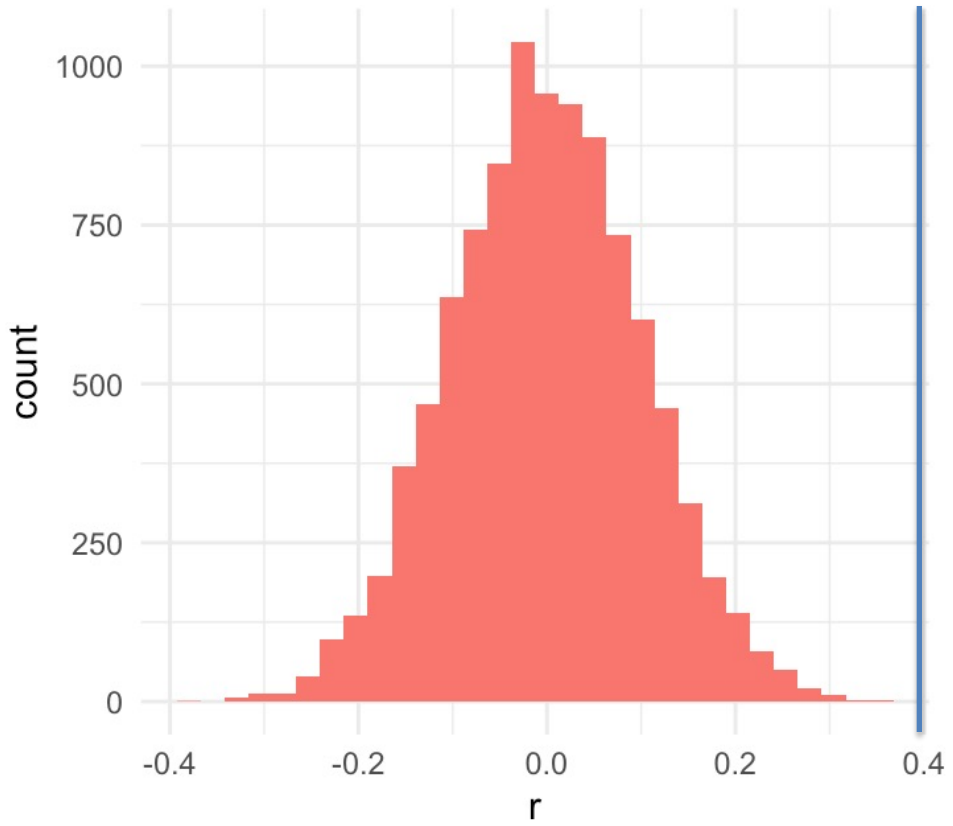
 If real data looks like shuffled data, then you cant tell apart signal/noise



Instead of making 20 graphs and doing this by eye. We will generate a histogram of statistics measuring structure, and compare our own.



With a clearer signal, you can discern real from shuffled data.


1000
750
500
250
0
count
-0.4
-0.2
0.0
0.2
0.4
r

Basics › NHST

Null hypothesis significance testing

(1) Define a ‘statistic’ that measures some structure you want to argue exists in the data.

(2) Define ‘null’ hypothesis (Ho): a model of your data if the structure of interest didn’t exist.

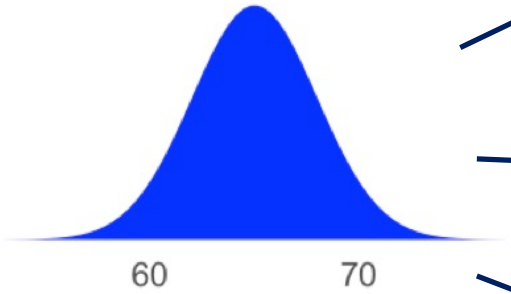
(3) Derive (analytically or numerically) the sampling distribution of the statistic of interest under Ho.

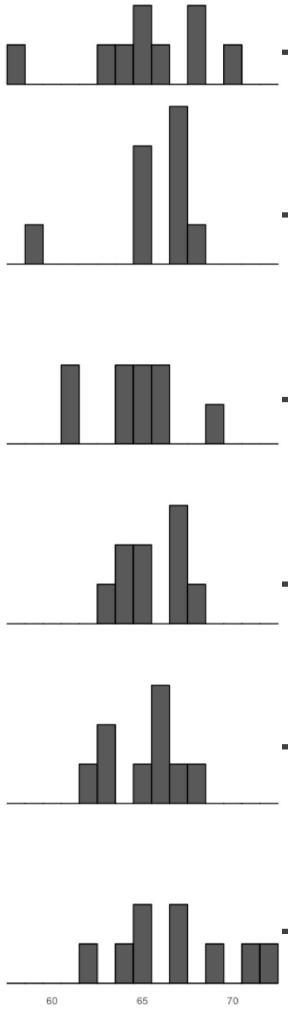
(4) What is the probability of seeing a statistic at least as extreme as the one you saw under Ho? (p-value)

(5) Reject Ho if your p-value is lower than acceptable Type I error rate (alpha)

(Mope if your p-value is higher than alpha)

Theoretical population
Statistical model
Null hypothesis





mean(x) = 65.3

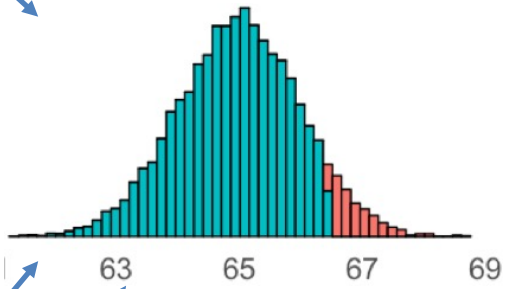
mean(x) = 65.5

mean(x) = 65.4

mean(x) = 64.4

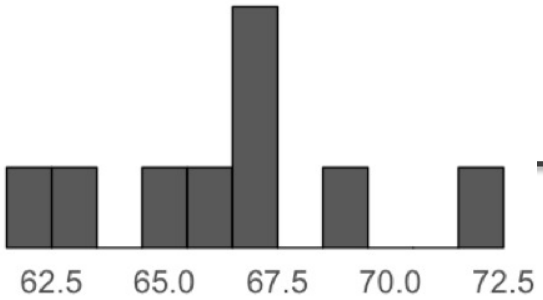
mean(x) = 65.5

mean(x) = 66.8



Null Hypothesis testing:
What is the probability that a random sample from the null model will have a statistic at least as extreme as the one from our data?
Here: 0.06
This is the *one-tailed* p-value.

Our data
(sample of 9 female heights, in inches)



A statistic
(arithmetic mean)

mean(x) = 66.44

Basics › NHST

Null hypothesis significance testing

(1) Define a 'statistic' that measures some structure you want to argue exists in the data.

(2) Define 'null' hypothesis (Ho): a model of your data if the structure of interest didn't exist.

(3) Derive (analytically or *numerically*) the sampling distribution of the statistic of interest under Ho.

(4) What is the probability of seeing a statistic at least as extreme as the one you saw under Ho? (p-value)

(5) Reject Ho if your p-value is lower than acceptable Type I error rate (alpha)

(Mope if your p-value is higher than alpha)

Resampling: Randomization

- Identify structure of interest
  e.g., different means across groups
- Define test statistic that measures structure
  e.g., mean(x1)-mean(x2)
- Resample data while destroying structure of interest, but preserving all other structure
  e.g. preserve ns, destroy mean difference.
  - Often done by reshuffling labels, somehow
  - *Permutation*: Ho distribution built by considering every label permutation
  - *Randomization*: Ho dist. built by considering some (large number of) random shuffles of labels

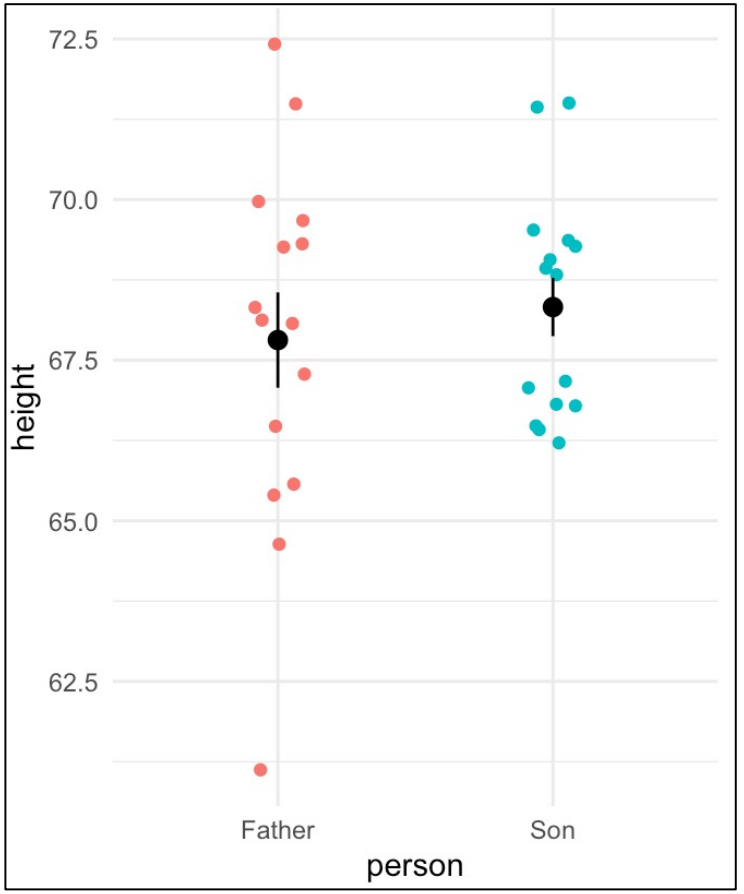
Randomization tests: diff. in means

Pearson Father-Son height data (15 Fathers, 15 Sons, unrelated)

```
> dat %>% print(n=40)
# A tibble: 30 x 2
   person height
   <chr>   <dbl>
 1 Father   69.3
 2 Father   66.5
 3 Father   65.4
 4 Father   70
 5 Father   61.1
 6 Father   64.6
 7 Father   68.1
 8 Father   67.3
 9 Father   65.6
10 Father   72.4
11 Father   71.5
12 Father   68.3
13 Father   69.7
14 Father   68.1
15 Father   69.3
16 Son      67.1
17 Son      69.3
18 Son      66.5
19 Son      68.8
20 Son      66.8
21 Son      66.4
22 Son      69.1
23 Son      69.5
24 Son      67.2
25 Son      71.5
26 Son      71.4
27 Son      66.2
28 Son      69.4
29 Son      66.8
30 Son      68.9
```

```
> glimpse(dat)
Rows: 30
Columns: 2
$ person <chr> "Father", "Father", "Fathe…
$ height <dbl> 69.3, 66.5, 65.4, 70.0, 61…
```

```
dat %>%
  ggplot(aes(x=person, y=height, color=person))+
  geom_point(position = position_jitter(width = 0.1))+
  stat_summary(fun.data = mean_se,
               geom='pointrange',
               color='black')+
  theme_minimal()
```



Randomization tests: statistic

Pearson Father-Son height data (15 Fathers, 15 Sons, unrelated)

```
> dat %>% print(n=40)
# A tibble: 30 x 2
   person height
   <chr>   <dbl>
 1 Father   69.3
 2 Father   66.5
 3 Father   65.4
 4 Father   70
 5 Father   61.1
 6 Father   64.6
 7 Father   68.1
 8 Father   67.3
 9 Father   65.6
10 Father   72.4
11 Father   71.5
12 Father   68.3
13 Father   69.7
14 Father   68.1
15 Father   69.3
16 Son      67.1
17 Son      69.3
18 Son      66.5
19 Son      68.8
20 Son      66.8
21 Son      66.4
22 Son      69.1
23 Son      69.5
24 Son      67.2
25 Son      71.5
26 Son      71.4
27 Son      66.2
28 Son      69.4
29 Son      66.8
30 Son      68.9
```

Test statistic that measures structure of interest (here: difference in means)

```
statistic = function(data){
  data %>%
    group_by(person) %>%
    summarize(m = mean(height)) %>%
    summarize(d = m[person == 'Son'] -
                  m[person == 'Father']) %>%
    pull(d)
}
```

```
> (my_stat = statistic(dat))
[1] 0.5133333
```

gy

Randomization tests: shuffle

Pearson Father-Son height data (15 Fathers, 15 Sons, unrelated)

```
> dat %>% print(n=40)
# A tibble: 30 x 2
   person height
   <chr>   <dbl>
 1 Father   69.3
 2 Father   66.5
 3 Father   65.4
 4 Father   70
 5 Father   61.1
 6 Father   64.6
 7 Father   68.1
 8 Father   67.3
 9 Father   65.6
10 Father   72.4
11 Father   71.5
12 Father   68.3
13 Father   69.7
14 Father   68.1
15 Father   69.3
16 Son      67.1
17 Son      69.3
18 Son      66.5
19 Son      68.8
20 Son      66.8
21 Son      66.4
22 Son      69.1
23 Son      69.5
24 Son      67.2
25 Son      71.5
26 Son      71.4
27 Son      66.2
28 Son      69.4
29 Son      66.8
30 Son      68.9
```

Randomize/shuffle data to destroy structure of interest (here: *shuffle* group

```
shuffle = function(data){
  data %>%
    mutate(person = sample(person, n(), replace=F))
}
```

Note: Sample *without* replacement -> shuffle labels.

```
> statistic(shuffle(dat))
[1] 0.9933333
> statistic(shuffle(dat))
[1] 0.8866667
> statistic(shuffle(dat))
[1] -0.5
```

```
> shuffle(dat) %>% print(n=40)
# A tibble: 30 x 2
   person height
   <chr>   <dbl>
 1 Father   69.3
 2 Son      66.5
 3 Father   65.4
 4 Son      70
 5 Son      61.1
 6 Father   64.6
 7 Son      68.1
 8 Son      67.3
 9 Father   65.6
10 Father   72.4
11 Father   71.5
12 Father   68.3
13 Father   69.7
14 Son      68.1
15 Son      69.3
16 Son      67.1
17 Son      69.3
18 Son      66.5
19 Father   68.8
20 Father   66.8
21 Son      66.4
22 Father   69.1
23 Father   69.5
24 Father   67.2
25 Son      71.5
26 Son      71.4
27 Son      66.2
28 Father   69.4
29 Son      66.8
30 Father   68.9
```

gy

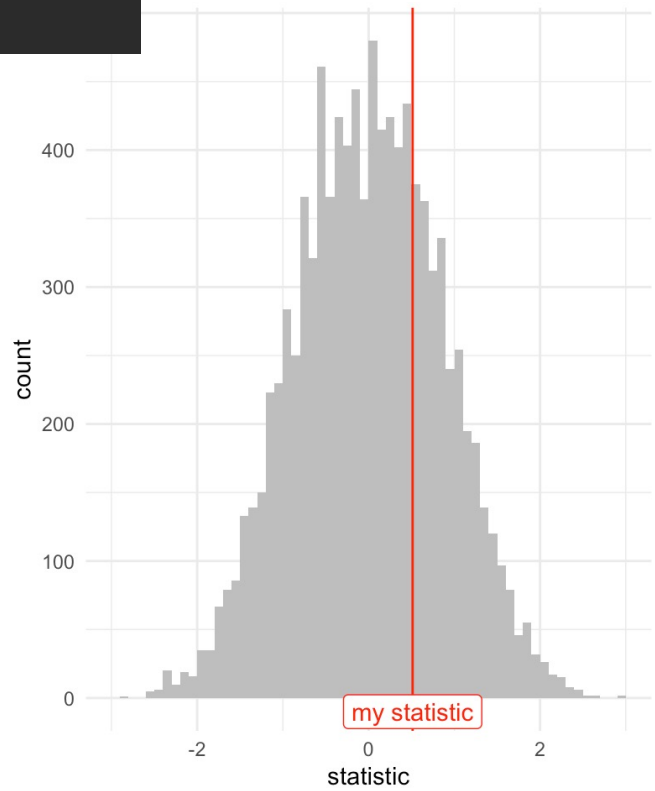
Randomization tests: build Ho distribution

Pearson Father-Son height data (15 Fathers, 15 Sons, unrelated)

```
> dat %>% print(n=40)
# A tibble: 30 x 2
   person height
   <chr>   <dbl>
 1 Father   69.3
 2 Father   66.5
 3 Father   65.4
 4 Father   70
 5 Father   61.1
 6 Father   64.6
 7 Father   68.1
 8 Father   67.3
 9 Father   65.6
10 Father   72.4
11 Father   71.5
12 Father   68.3
13 Father   69.7
14 Father   68.1
15 Father   69.3
16 Son      67.1
17 Son      69.3
18 Son      66.5
19 Son      68.8
20 Son      66.8
21 Son      66.4
22 Son      69.1
23 Son      69.5
24 Son      67.2
25 Son      71.5
26 Son      71.4
27 Son      66.2
28 Son      69.4
29 Son      66.8
30 Son      68.9
```

Generate null hypothesis (H0) samples of test statistic by computing it on shuffled data.

```
n_shuffles = 10000
null_stat = rep(NA, n_shuffles)
for(i in 1:n_shuffles){
  null_stat[i] = statistic(shuffle(dat))
}
```

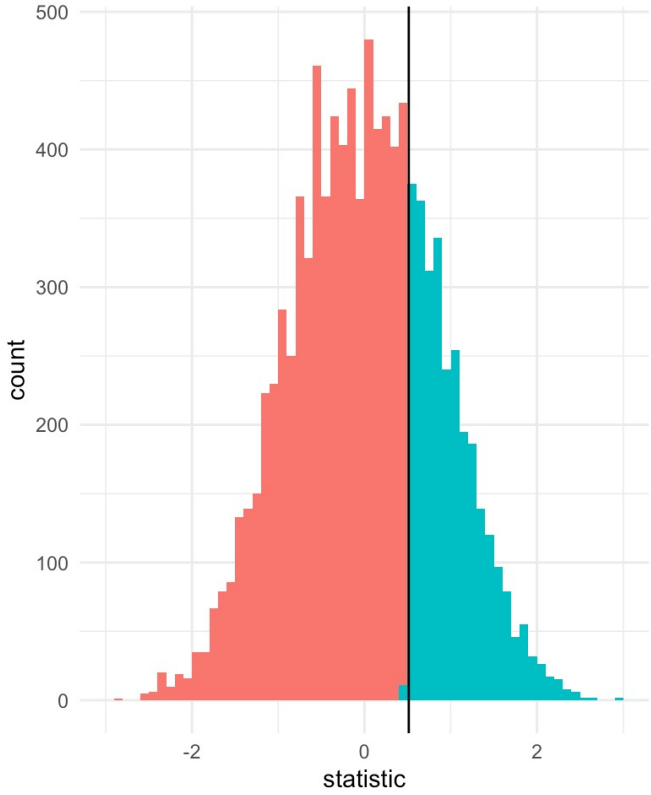


Randomization tests: calculate p value

Pearson Father-Son height data (15 Fathers, 15 Sons, unrelated)

```
> dat %>% print(n=40)
# A tibble: 30 x 2
   person height
   <chr>   <dbl>
 1 Father   69.3
 2 Father   66.5
 3 Father   65.4
 4 Father   70
 5 Father   61.1
 6 Father   64.6
 7 Father   68.1
 8 Father   67.3
 9 Father   65.6
10 Father   72.4
11 Father   71.5
12 Father   68.3
13 Father   69.7
14 Father   68.1
15 Father   69.3
16 Son      67.1
17 Son      69.3
18 Son      66.5
19 Son      68.8
20 Son      66.8
21 Son      66.4
22 Son      69.1
23 Son      69.5
24 Son      67.2
25 Son      71.5
26 Son      71.4
27 Son      66.2
28 Son      69.4
29 Son      66.8
30 Son      68.9
```

P value numerically calculated from H0 samples.




```
(sum(null_stat >= my_stat) + 1) / (n_shuffles + 2)
```

Note:

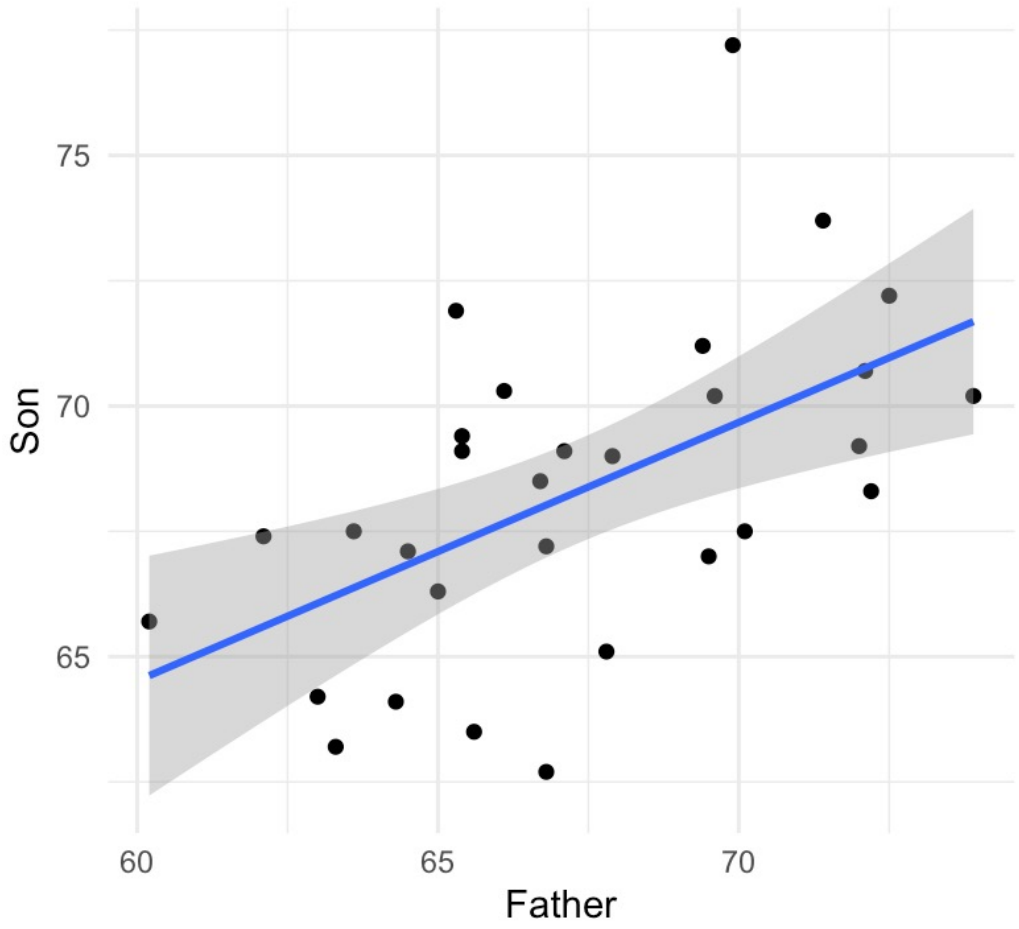
Here: one-tailed -- upper tail only

(x2 for second tail on symmetric stats)

Smoothed by assuming 2 extra observations to avoid p=0

You must respect other structure

```
> dat
# A tibble: 15 x 3
   Father   Son  pair
    <dbl> <dbl> <int>
 1   67.5  66.1     1
 2   64.1  66.1     2
 3   62.5  64       3
 4   68.1  69.1     4
 5   70.2  71       5
 6   72.6  76.8     6
 7   65.9  71.4     7
 8   67.2  70.3     8
 9   68.2  72       9
10   72.4  76.4    10
11   65.4  69.1    11
12   64.9  70.5    12
13   70.2  61.2    13
14   71.9  68.4    14
15   66.8  68.3    15
```



Paired data statistic

```
> dat
# A tibble: 60 x 3
    pair person height
   <int> <chr>   <dbl>
 1     1 Father   65.4
 2     1 Son      69.1
 3     2 Father   65.6
 4     2 Son      63.5
 5     3 Father   63.3
 6     3 Son      63.2
 7     4 Father   63.6
 8     4 Son      67.5
 9     5 Father   66.1
10     5 Son      70.3
# … with 50 more rows
```

```
statistic = function(data){
  data %>%
    pivot_wider(id_cols = pair,
                names_from = person,
                values_from = height) %>%
    mutate(delta = Son-Father) %>%
    summarize(md = mean(delta)) %>%
    pull(md)
}
```

```
# A tibble: 30 x 3
    pair Father   Son
   <int>  <dbl> <dbl>
 1     1   65.4  69.1
 2     2   65.6  63.5
 3     3   63.3  63.2
 4     4   63.6  67.5
 5     5   66.1  70.3
 6     6   66.7  68.5
 7     7   69.9  77.2
 8     8   65.4  69.4
 9     9   72    69.2
10    10   65.3  71.9
```

```
> (my_stat = statistic(dat))
[1] 0.9733333
```

Paired data shuffling

```
> dat
# A tibble: 60 x 3
    pair person height
   <int> <chr>   <dbl>
 1     1 Father   65.4
 2     1 Son      69.1
 3     2 Father   65.6
 4     2 Son      63.5
 5     3 Father   63.3
 6     3 Son      63.2
 7     4 Father   63.6
 8     4 Son      67.5
 9     5 Father   66.1
10     5 Son      70.3
# … with 50 more rows
```

```
shuffle = function(data){
  dat %>%
    group_by(pair) %>%
    mutate(person = sample(person, n(), replace=F)) %>%
    ungroup()
}
```

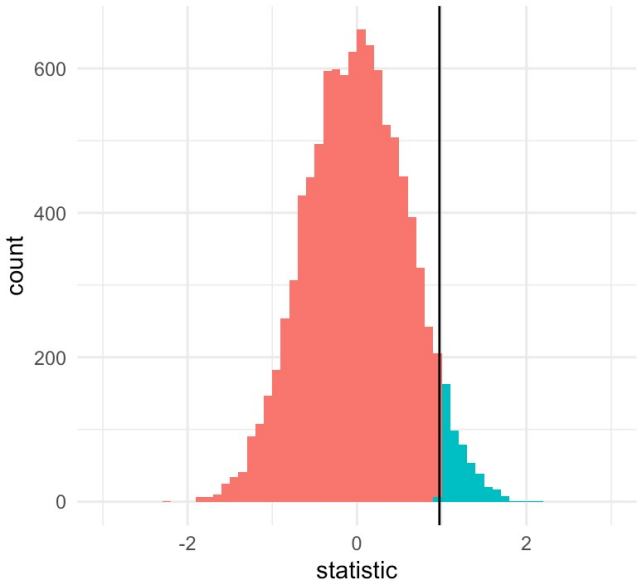
Note: Shuffling *within* father-son pairs, to preserve that structure

```
> shuffle(dat)
# A tibble: 60 x 3
    pair person height
   <int> <chr>   <dbl>
 1     1 Father   65.4
 2     1 Son      69.1
 3     2 Son      65.6
 4     2 Father   63.5
 5     3 Son      63.3
 6     3 Father   63.2
 7     4 Father   63.6
 8     4 Son      67.5
 9     5 Father   66.1
10     5 Son      70.3
# … with 50 more rows
```

Paired data Ho distribution and p value

```
> dat
# A tibble: 60 x 3
    pair person height
   <int> <chr>   <dbl>
 1     1 Father   65.4
 2     1 Son      69.1
 3     2 Father   65.6
 4     2 Son      63.5
 5     3 Father   63.3
 6     3 Son      63.2
 7     4 Father   63.6
 8     4 Son      67.5
 9     5 Father   66.1
10     5 Son      70.3
# … with 50 more rows
```

```
n_shuffles = 10000
null_stat = rep(NA, n_shuffles)
for(i in 1:n_shuffles){
  null_stat[i] = statistic(shuffle(dat))
}
```




```
> (sum(null_stat >= my_stat) + 1) / (n_shuffles + 2)
[1] 0.0539892
```

Note:

One-tailed (upper tail only)

Smoothed by assuming 2 extra observations to avoid p=0

Randomization

- Why are we doing this? Why don't we just do a t-test?
  - Sometimes: We don't trust analytical model assumptions.
    e.g., chi-squared test…
  - Usually: Our structure is most effectively measured by some weird statistic for which we do not have an analytical null distribution
    e.g., median? trimmed-mean? Ratio of variances? Difference in kurtosis? Co-clustering rate from some weird clustering algorithm? Whatever.
  - Really general procedure: there is one test.
    You just have to carefully think through
    how to measure the structure of interest
    how to define the null you want to compare against

Randomization tests: diff. in means

Test statistic that measures structure of interest (here: difference in means)

```
statistic = function(dat){with(data=dat, mean(x[y==1])-mean(x[y==0]))}
```

Obtain statistic on your real data

```
statistic(dat)
```

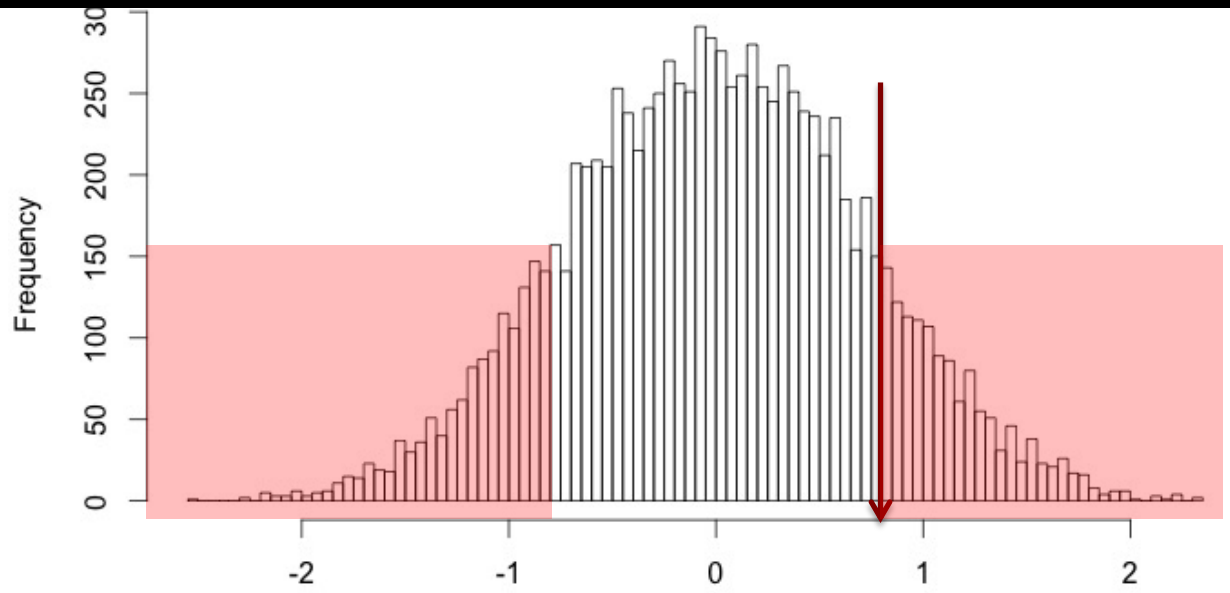
```
[1] 0.8049
```

Randomize/shuffle data to destroy structure of interest (here: *shuffle* group labels)

```
dat.shuffle = function(dat){data.frame(x=dat$x, y=sample(dat$y, length(dat$y), replace=F))}
```

Generate null hypothesis (H0) samples of test statistic by computing it on shuffled data.

```
H0.samps = replicate(K, statistic(dat.shuffle(dat)))
```



P value numerically calculated from H0 samples.

```
p.val = (sum(abs(H0.samps)>=statistic(dat))+1)/(length(H0.samps)+2)
```

```
[1] 0.2605
```

Randomization tests: slope of line

Test statistic that measures structure of interest (here: slope of regression line)

```
statistic = function(dat){with(data=dat, coefficients(lm(x~y))[2])}
```

Obtain statistic on your real data

```
statistic(dat)
```

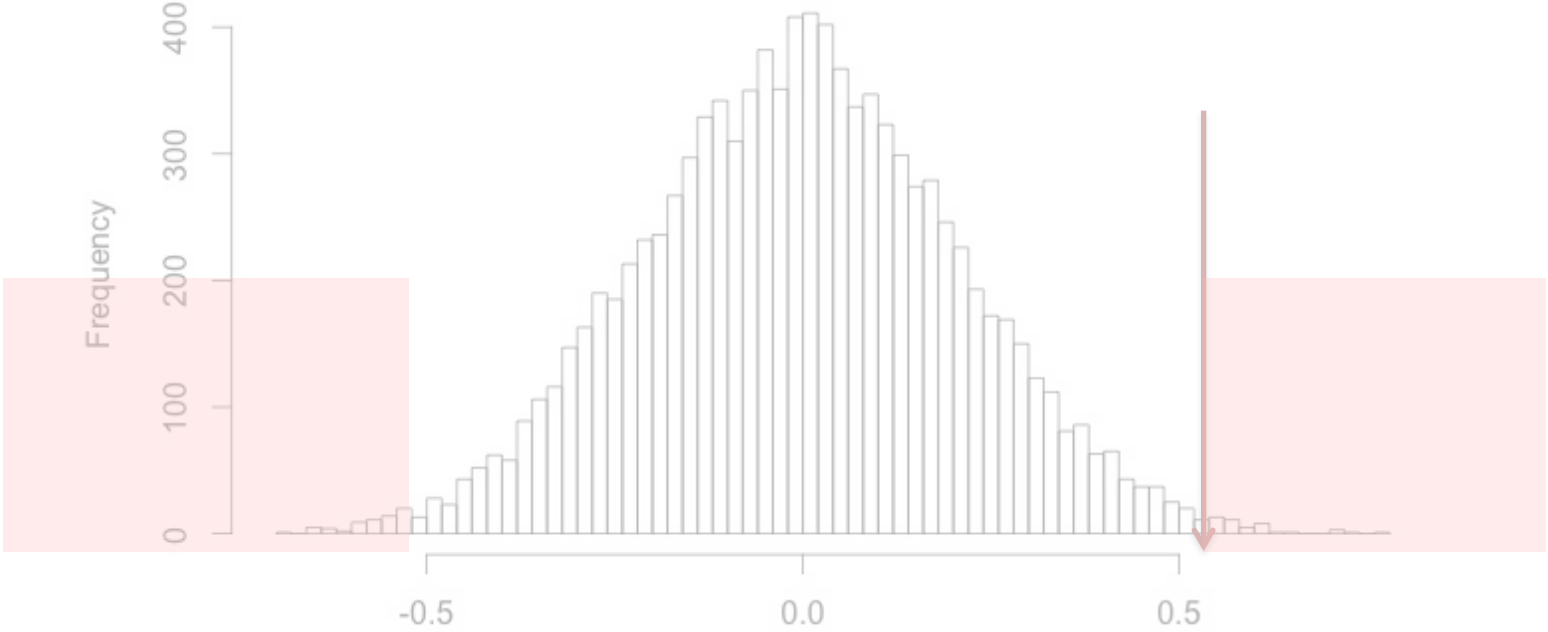
```
[1] 0.5323
```

Randomize/shuffle data to destroy structure of interest (here: shuffle y values)

```
dat.shuffle = function(dat){data.frame(x=dat$x, y=sample(dat$y, length(dat$y), replace=F))}
```

Generate null hypothesis (H0) samples of test statistic by computing it on shuffled data.

```
H0.samps = replicate(K, statistic(dat.shuffle(dat)))
```




```
p.val = (sum(abs(H0.samps)>=statistic(dat))+1)/(length(H0.samps)+2)
```

```
[1] 0.2605
```

Randomization tests: diff in variance

Test statistic that measures structure of interest (here: difference in variances)

```
statistic = function(dat){with(data=dat, log10(var(x[y==1])/var(x[y==0])))}
```

Obtain statistic on your real data

```
statistic(dat)
```

```
[1] 1.18
```

Randomize/shuffle data to destroy structure of interest (here: shuffle group labels)

```
dat.shuffle = function(dat){data.frame(x=dat$x, y=sample(dat$y, length(dat$y), replace=F))}
```

Generate null hypothesis (H0) samples of test statistic by computing it on shuffled data.

```
H0.samps = replicate(K, statistic(dat.shuffle(dat)))
```

But wait – we destroyed all the structure, not just the structure of interest!

```
p.val = (sum(abs(H0.samps)>=statistic(dat))+1)/(length(H0.samps)+2)
```

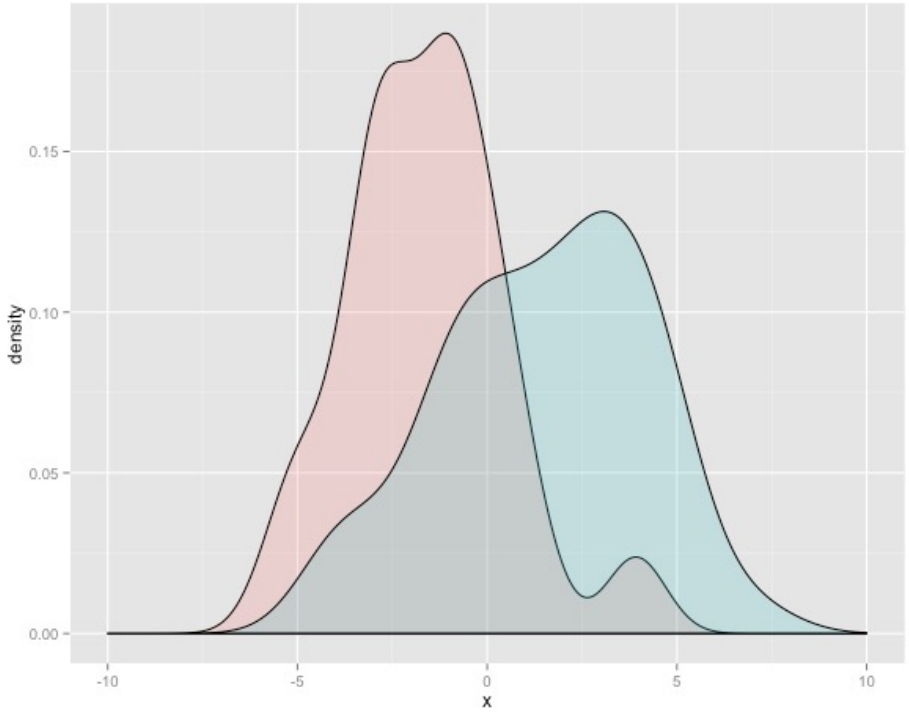
```
[1] 1e-04
```

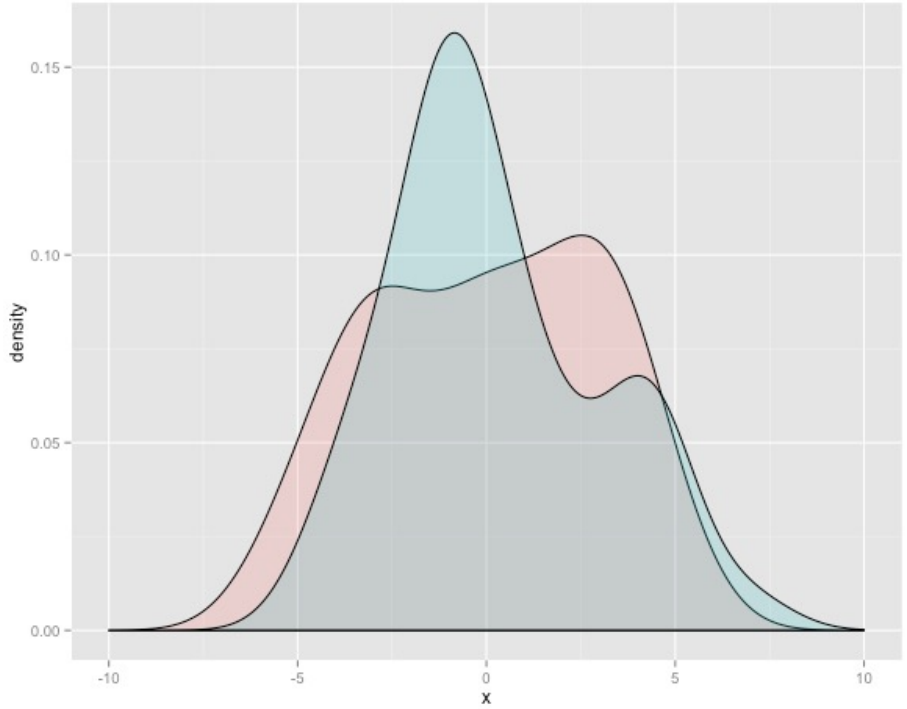
Preserve other structure.

Test statistic that measures structure of interest (here: difference in variances)

Randomize/shuffle data to destroy structure of interest (here: shuffle group labels)

```
dat.shuffle = function(dat){data.frame(x=dat$x, y=sample(dat$y, length(dat$y)))}
```





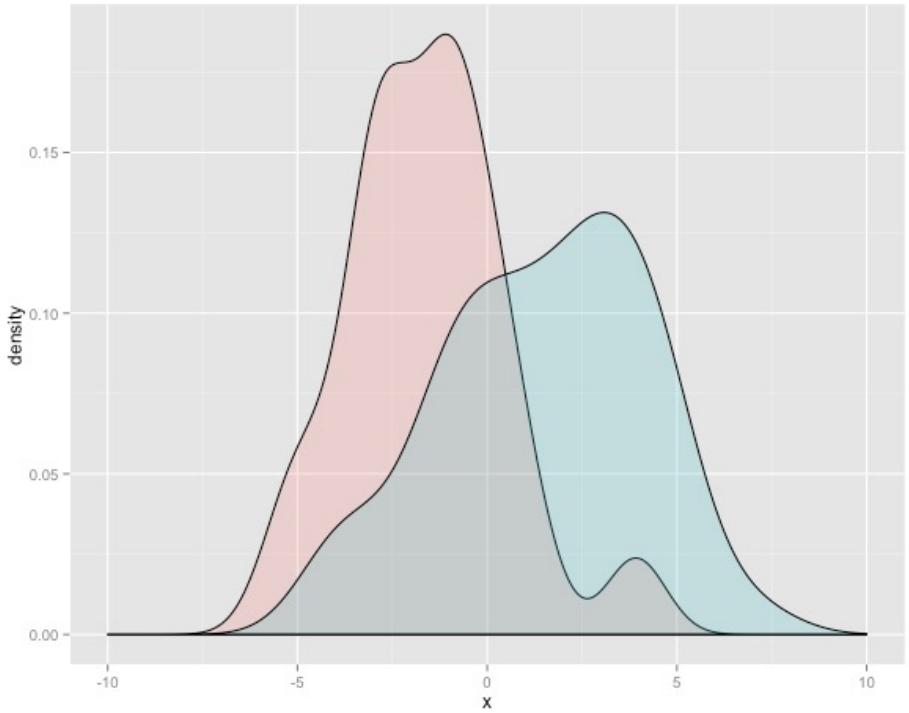
Resampled without preserving means!

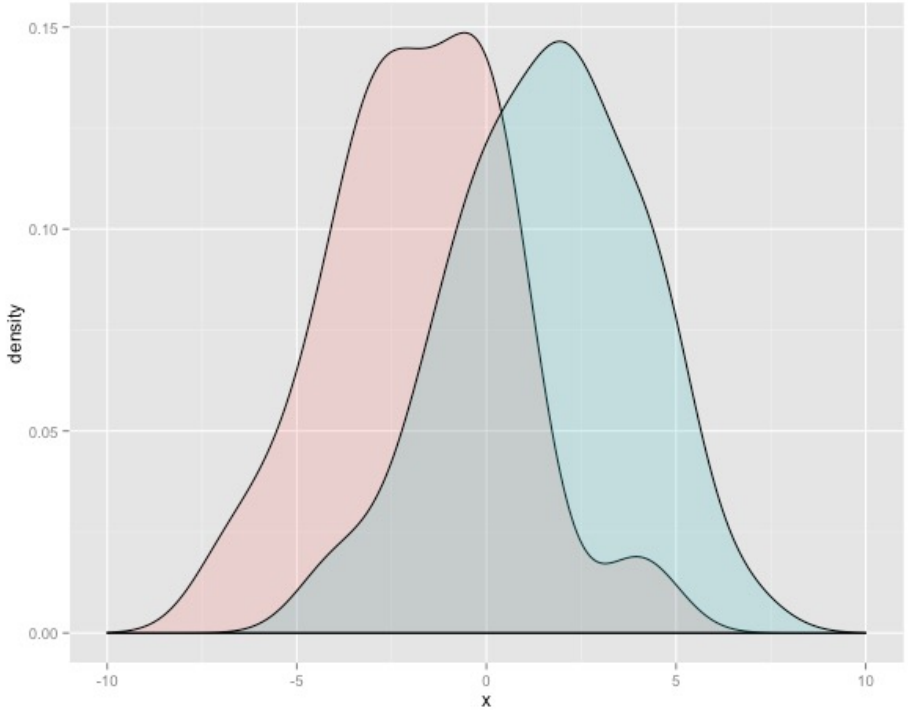
Preserve other structure.

Test statistic that measures structure of interest (here: difference in variances)

Randomize/shuffle data to destroy structure of interest while preserving other structure! (Here: shuffle residuals)

```
dat$r = residuals(lm(data=dat, x~as.factor(y)))
dat$m = dat$x-dat$r
dat.shuffle = function(dat){data.frame(
                                        x=dat$m+sample(dat$r,length(dat$r)),
                                        y=dat$y)}
```

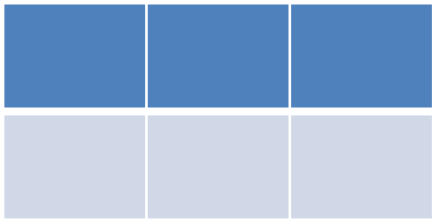




Preserved mean difference!

Randomization: ANOVA

- How do we shuffle an ANOVA?
- It depends on our null hypothesis.
  - Ho.a: All cells have the same mean, no effect at all
  - Ho.b: Main effect of A, but not B, and no interaction.
  - Ho.c: Two main effects, but no interaction.
  - Ho.d: Effect of covariate, but no effect of factors.
  - Resampling has to preserve some structure, while destroying other structure.



How to test the following?

1) We perform a robust regression (least-trimmed-squares), and get some slope out. How do we tell if it is significantly greater than we would expect under the null of no relationship?
2) 2 groups with unequal means and variances. We want to know if they have different skew.
3) 30 countries of varying mean wealth. We have 1000 randomly sampled people from each country. We want to know if there is greater inequality (variance) of wealth within a country if that country has a higher average wealth. (we want to deal with log-wealth).
4) We find that when people are listing words off the top of their heads sequential words tend to be more semantically related than those that are further apart. How would you test if this is significant?
5) We use a clustering algorithm to find 'types' of people based on test performance. It seems that these clusters effectively cluster autistic people together. How do we figure out if this is the case above chance?

Randomization: How many Ho samples can we get?

- How many Ho samples can we get?
- Number of permutations gives us upper bound for a given shuffling scheme.

- E.g. we have two groups of 5, how many different shuffled labelings can we have?

Randomization: How many Ho samples can we get?

- How many Ho samples can we get?
- Number of permutation gives us upper bound for a given shuffling scheme.
- E.g. we have two groups of 5, how many different shuffled labelings can we have?
  - 10 choose 5 = 10! / 5!5! = 252
  - So, no matter how many times we randomize, we aren't going to get more than 252 unique Ho test statistic samples.
  - If sample sizes are small, this is worth considering, and explicitly doing permutation tests is a better option.
- 2 groups of 10? 184756; no longer matters.

Randomization/Permutation/Shuffling tests.

1) Identify structure you want to test.
2) Define a statistic that measures this structure.
3) Shuffle (resampling without replacement!) the data in a way that disrupts this structure (capturing the null hypothesis) without disrupting other structure (otherwise you are testing a different null hypothesis).
4) Compute your statistic on lots of shuffled datasets, figure out if your statistic on the real (unshuffled) dataset is extreme relative to what is expected under the Ho shufflings.

Beware:

- All of this requires that you be thoughtful, rather than following simple prescriptions in your analysis.
- The number of data permutations puts a limit on the possible number of Ho samples. Use caution with small sample sizes.
- Elaborate procedures to deal with extreme value statistics.

Resampling: Overview

- Randomization / Permutation / Shuffling:
  - Define a statistic that measures structure of interest
  - Resample so as to destroy the structure of interest
  - Calculate statistic on shuffled samples
  - Distribution of shuffled statistics is the null hypothesis sampling distribution of the statistic, compare statistic on real data to this
- **Bootstrapping:**
  - Draw more samples like the current one
  - Run some estimator on those samples
  - To build a sampling distribution of an estimate
  - This is useful for confidence intervals, etc.

Bootstrap: Logic

- Future data will be like the current data
- We can generate more samples of our data by resampling from the PDF made up of our data.

```
x
```

```
[100] -0.81057568 -3.75588056 -0.77649265 -2.71176279 -0.72528754 -1.79951633  0.28982537 -1.86105795 -0.08463999  0.08801584  0.42490192 -0.32787760  1.40340032
0.35678602 -1.41562830 -1.04574622  0.09898905  1.05035238 -2.16351814 -2.54987788  0.60558522  0.98092604 -0.61329110 -0.42499095 -0.75423395 -0.40166514
0.18090021 -0.57891140 -0.63355362  1.42385112  0.19384376 -1.50328405  1.16187994 -2.61733233  1.33129141 -0.63628871 -0.45647740 -5.07116647 -1.74510401 -
1.41756240  0.19368629 -0.09093373 -0.97658404 -6.49857275 -1.61333570  0.22196662 -0.24209335  0.75823578 -0.31382485  1.22549832  2.26054057  0.18598882
1.78827723  1.21081248 -2.81913736  1.97343288 -1.28652368  1.54594865  0.93246801  0.23824970  0.35839401  0.57989193 -4.64778399 -1.25622966 -4.42018876
0.20398443 -1.23291749 -0.44188558  0.61962049  0.81900205 -1.47554994  0.03681568 -2.06985538 -0.26888997 -1.32263459 -1.31015657  0.49810722  0.08016335
0.73409104 -0.37473450 -3.96311430 -3.79768621  0.06302523  1.16587685  1.47153014  0.54874491 -0.62163567 -1.88914946 -0.69980288  2.42535899 -1.59746565 -
6.15922478  0.40459299  1.05146166  0.47986380  0.44199065  0.91806090 -0.49377271  1.20637696  0.03993493
```

```
x.new = function(){sample(x, n, replace=TRUE)}
```

Sample from the data

Same sample size

With replacement!

- Each resampled sample is a glimpse at what other data would look like, given current data.

Resampling: Bootstrap.

```
skew = function(x)(mean(((x-mean(x))/sd(x))^3))          <- Estimator
```

```
skew(x)                                                  [1] -1.218723
```

- Generate new resampled samples of x, for each one obtain an estimate

```
x.new = function(){sample(x, n, replace=TRUE)}           <- Resample function
```

```
skew(x.new())                [1] -0.876                  <- One resampled skew estimate
```

```
resampled.skews = replicate(10000, skew(x.new()))
                                                         10000 resampled skew estimates
```

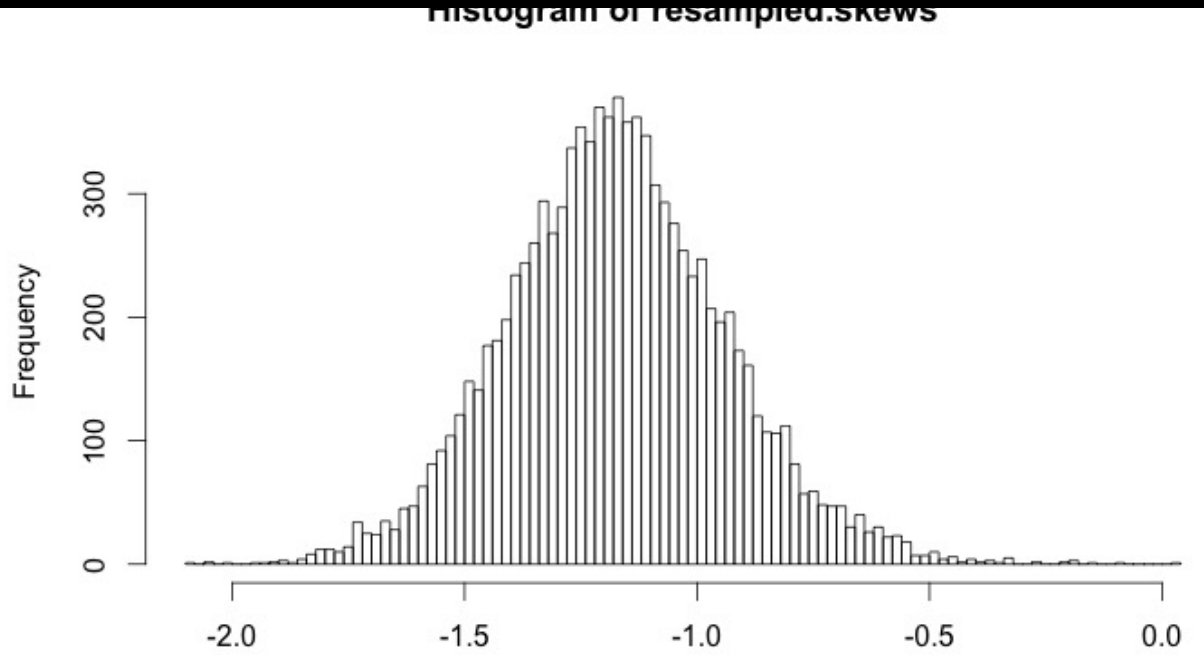
Resampling: Bootstrap.

- Make an estimator (here, skew)

```
skew = function(x)(mean(((x-mean(x))/sd(x))^3))          <- Estimator
skew(x)                                                  [1] -1.218723
```

- Generate new resampled samples of x, for each one obtain an estimate

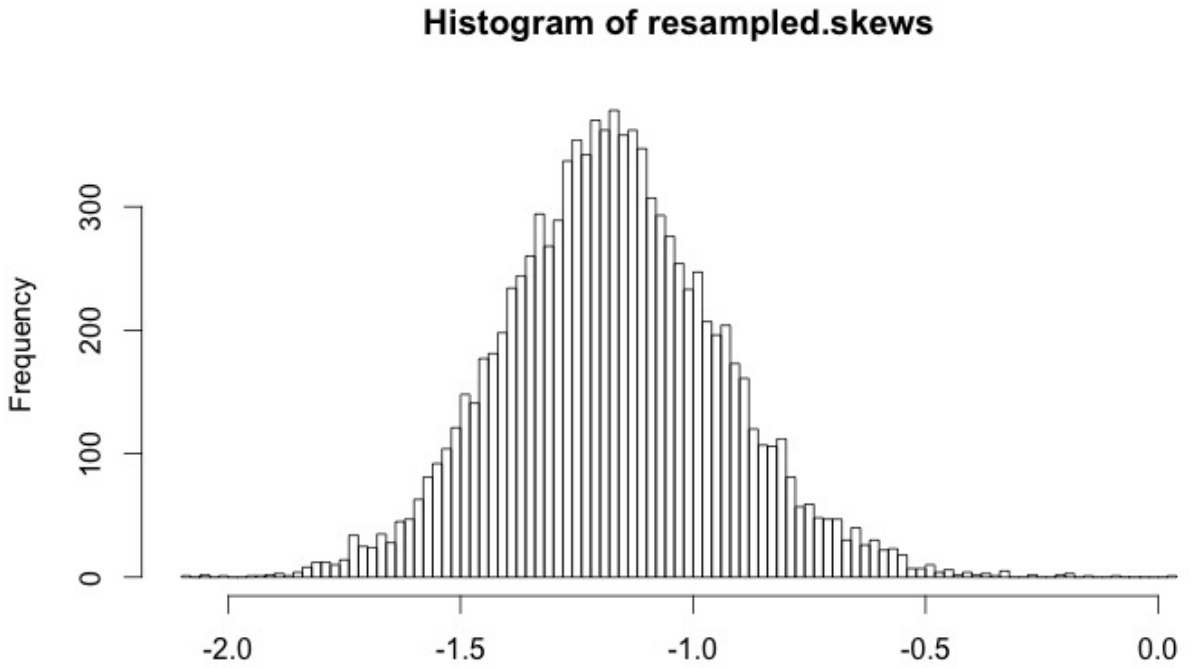
```
x.new = function(){sample(x, n, replace=TRUE)}           <- Resample function
resampled.skews = replicate(10000, skew(x.new()))
```




```
hist(resampled.skews)
```

Resampling: Bootstrap.

- Make an estimator (here, skew)
- Generate new resampled samples of x, for each one obtain an estimate
- This is our guess as to the sampling distribution of the estimator. So we can use it to do inference




```
hist(resampled.skews)
```

Resampling: Bootstrap.

- Make an estimator (here, skew)
- Generate new resampled samples of x, for each one obtain an estimate
- This is our guess as to the sampling distribution of the estimator. So we can use it to do inference
- Confidence intervals:

```
quantile(resampled.skews, c(0.025, 0.975))
```

```
     2.5%      97.5%
-1.625407 -0.680099
```

- Is skew less than 0 (2-tailed p.val for null of >0)

```
2*sum(resampled.skews>0)/length(resampled.skews)
```

```
0.0002
```

Helpful to 'smooth' counts

```
2*(sum(resampled.skews>0)+1)/(length(resampled.skews)+2)
```

```
0.00039992
```

Resampling: Bootstrapping.

Goal: Find sampling distribution of some statistic

Define an estimator for some statistic of interest

```
statistic = function(x)(...)
```

Define resampling function: draw samples of size length(data) from the data, with replacement.

```
x.new = function(){sample(x, length(x), replace=TRUE)}
```

Draw K samples of statistic estimated from resampled data. K should be large.

```
res.stats = replicate(K, statistic(x.new()))
```

Use sampled statistic to define confidence intervals…

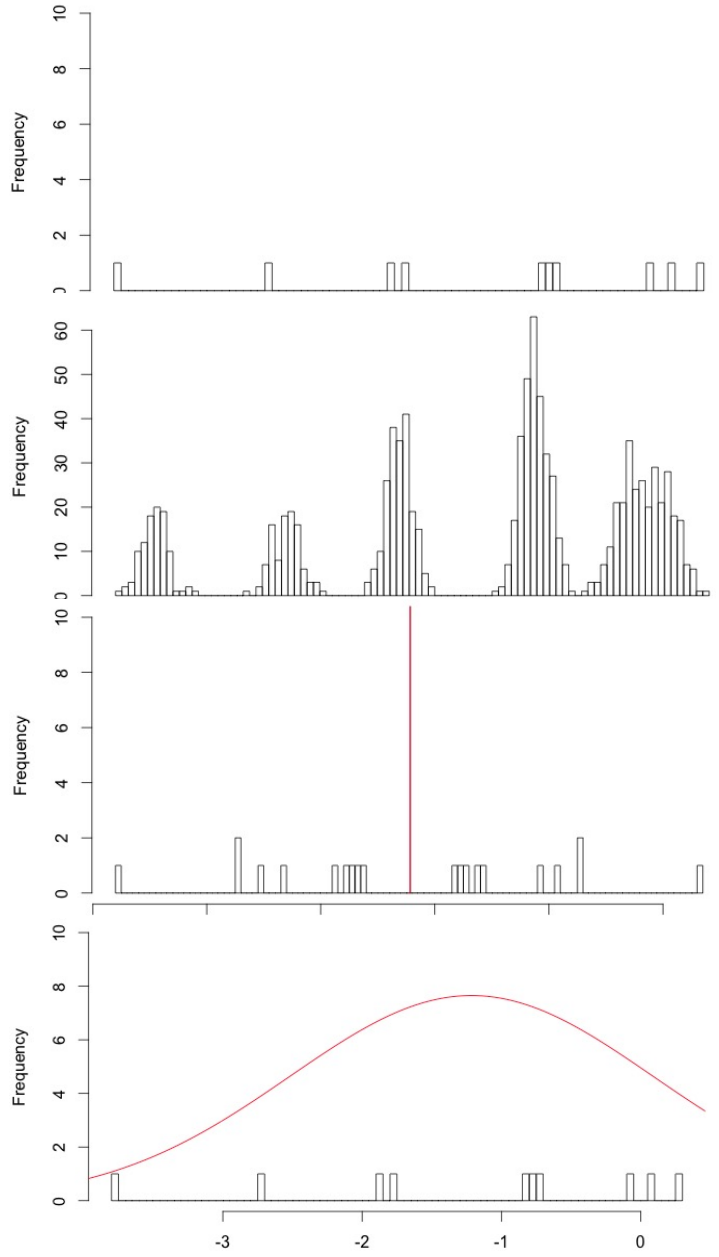
```
CI = quantile(res.stats, c(alpha/2, 1-alpha/2))
```

Use sampled statistic to obtain p.values.

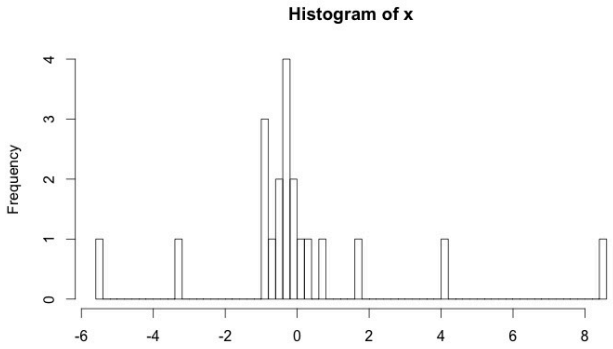
```
p.val = 2*(sum(res.stats >= H0.stat)+1)/(length(res.stats)+2)
```

Bootstrap variants

- **Case resampling:**
  Resample data exactly
  from current data
- **Smoothed bootstrap:**
  Resample data with some extra
  noise
- **Residual/Pivoted bootstrap:**
  Resample residuals (with
  assumed symmetry?)
- **Parametric bootstrap:**
  Resample data from some fitted
  distribution



Resampling: adding invariance / noise.




```
statistic = function(x)(sd(x))
statistic(x)
```
```
[1] 2.692358
```

Case resampling

```
x.new = function(){sample(x, n, TRUE)}
CI = quantile(replicate(K, statistic(x.new())), c(0.025, 0.975))
```
```
[2] 1.01 3.87
```

Smooth resampling

```
x.new = function(){sample(x, n, TRUE)+rnorm(n,0,sd(x)/10)}
CI = quantile(replicate(K, statistic(x.new())), c(0.025, 0.975))
```
```
[2] 0.98 3.92
```

Pivoted resampling

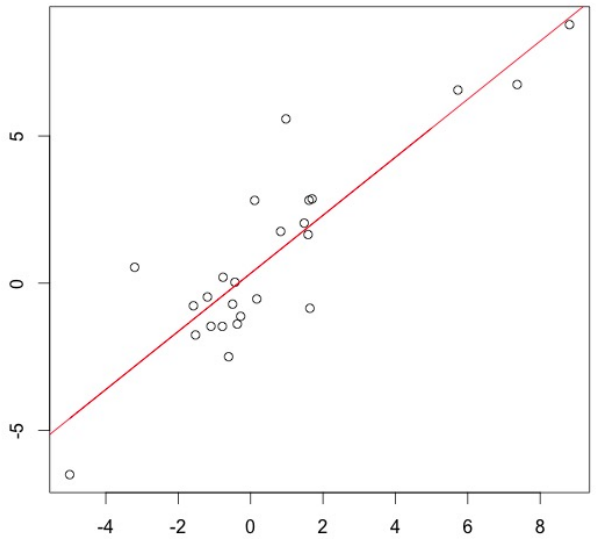
```
x.new = function(){sample(x-mean(x), n, TRUE)*sample(c(-1,1), n, TRUE)+mean(x)}
CI = quantile(replicate(K, statistic(x.new())), c(0.025, 0.975))
```
```
[2] 1.03 3.96
```

Parametric bootstrap

```
x.new = function(){rnorm(n,mean(x), sd(x))}
CI = quantile(replicate(K, statistic(x.new())), c(0.025, 0.975))
```
```
[2] 1.83 3.54
```

Parametric bootstrap generally not a good idea: removes most advantages of of resampling…

Resampling structured data.




```
B0 = function(dat){coefficients(lm(dat$y~dat$x))[1]}
```

```
B1 = function(dat){coefficients(lm(dat$y~dat$x))[2]}
```

```
B0(dat)                                    [1] 0.33
B1(dat)                                    [1] 0.99
```

How to do case resampling?

```
dat.new = function(dat){
                    data.frame(x=dat[sample(1:n, n, TRUE),'x'],
                               y=dat[sample(1:n, n, TRUE),'y'])}
```

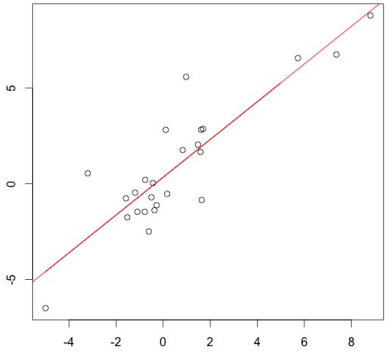
```
dat.new = function(dat){idx = sample(1:n, n, TRUE);
                        data.frame(x=dat[idx,'x'],
                                   y=dat[idx,'y'])}
```

Which option should we prefer, and why?

```
dat.new = function(dat){idx = sample(1:n, n, TRUE);
                        data.frame(x=dat[idx,'x'],
                                   y=dat[idx,'y'])}
```

This options preserves the relationship between x and y by resampling *pairs* of data points!
We really want this, otherwise we get samples from Null!

Resampling structured data.



Case resampling

```
B0 = function(dat){coefficients(lm(dat$y~dat$x))[1]}
```

```
B1 = function(dat){coefficients(lm(dat$y~dat$x))[2]}
```

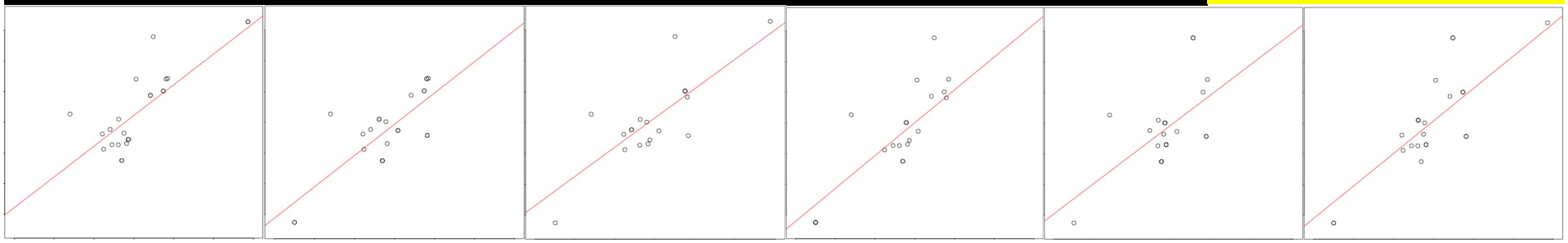
```
B0(dat)
```
```
[1] 0.33
```

```
B1(dat)
```
```
[1] 0.99
```

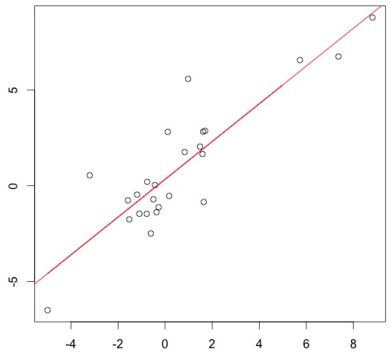
```
dat.new = function(dat){idx = sample(1:n, n, TRUE);
                        data.frame(x=dat[idx,'x'],
                                   y=dat[idx,'y'])}
```

```
B0.CI = quantile(replicate(K, B0(dat.new(dat))), c(0.025, 0.975))
```
```
[2] -0.30 1.03
```

```
B1.CI = quantile(replicate(K, B1(dat.new(dat))), c(0.025, 0.975))
```
```
[2] 0.78 1.22
```



Resampling structured data.




```
B0 = function(dat){coefficients(lm(dat$y~dat$x))[1]}
```

```
B1 = function(dat){coefficients(lm(dat$y~dat$x))[2]}
```

```
B0(dat)
```
```
[1] 0.33
```

```
B1(dat)
```
```
[1] 0.99
```

Case resampling

```
dat.new = function(dat){idx = sample(1:n, n, TRUE);
                        data.frame(x=dat[idx,'x'],
                                   y=dat[idx,'y'])}
```

```
B0.CI = quantile(replicate(K, B0(dat.new(dat))), c(0.025, 0.975))
```
```
[2] -0.30 1.03
```

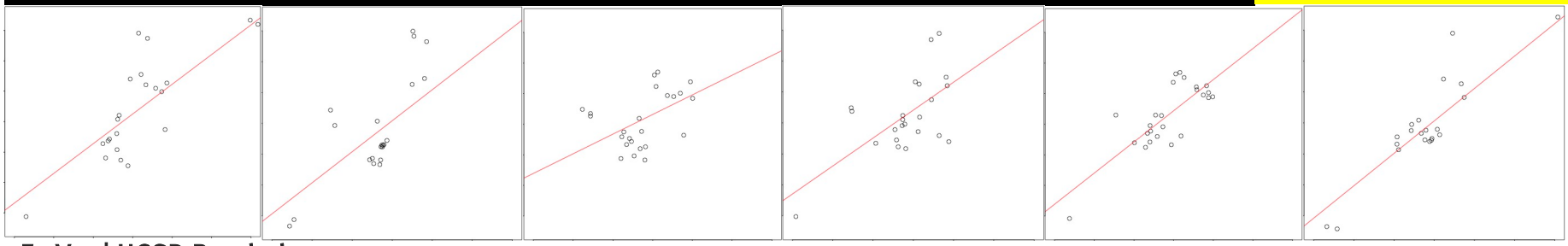
```
B1.CI = quantile(replicate(K, B1(dat.new(dat))), c(0.025, 0.975))
```
```
[2] 0.78 1.22
```

Smooth resampling

```
dat.new = function(dat){idx = sample(1:n, n, TRUE);
                        data.frame(x=(dat[idx,'x']+rnorm(n,0,sd(dat$x)/10)),
                                   y=(dat[idx,'y']+rnorm(n,0,sd(dat$y)/10)))}
```

```
B0.CI = quantile(replicate(K, B0(dat.new(dat))), c(0.025, 0.975))
```
```
[2] -0.31 1.07
```

```
B1.CI = quantile(replicate(K, B1(dat.new(dat))), c(0.025, 0.975))
```
```
[2] 0.76 1.21
```



Resampling: Benefits.

- Can build sampling distributions, estimate confidence intervals for arbitrary statistics and measures.
  - Consider: Skew, Kurtosis, etc.
  - Various measures that make sense scientifically, but are not convenient mathematically
- General purpose tool – no need to remember specific rules for specific problems!
- Does not make distributional assumptions – respects the data, and distributional assumptions can't be violated.

Resampling: Costs.

- Computationally expensive: requires lots of resampling, can take some time.
- Breaks down when sample sizes are small (bootstrap estimates are too narrow).
- Tends to be somewhat narrow-minded: if I haven't seen outliers before, I imagine they do not exist.
  - (Wild bootstrap, smooth bootstrap, and other techniques exist for adding uncertainty to resampling, but how do you know how much uncertainty to add?)
- Can get quite hairy for richly structured data that has various dependencies…

How many samples? How much data?

- How many bootstrap samples (K) should you use?
  - The more the merrier – standards increase with Moore's law
  - Minimum: if you want to specify probability with a precision of *u*, you should have at least 1/u bootstrap samples.
    E.g., 95% interval requires 0.025th quantile (precision of 0.001), requires K›=1000
  - I like K›=10,000. I use K=1,000 if I'm in a rush.
- How much data (n) do you need to bootstrap?
  - Number of possible unique samples W=choose(2n-1,n-1)

n	1	2	3	4	5	6	7	8	9	10	15
W	1	3	10	35	126	462	1716	6435	24310	92378	77558760

  - Bootstrapping makes sense if W ›› K; let's say n›=15 or 20
  - However, the reliability of bootstrapped CIs etc. depends on the statistic; the more the statistic measures extremes, the more data you need.

Resampling: Bootstrap

- Use case:
  - You want a confidence interval / estimate of precision of some statistic(s); however…
  - You want to avoid parametric assumptions, or the statistic has no analytical sampling distribution or likelihood function.
- Premise:
  - treat your data histogram as the population distribution.
  - Resample from your data *with replacement* to get bootstrapped samples *of the same size* to approximate the true sampling distribution of your statistic(s) of interest.
  - Build confidence intervals using that bootstrapped sampling distribution.
- Cautionary notes:
  - You should have a reasonably large sample to begin with.
  - Think carefully about resampling procedure for structured data.
  - Some statistics (those sensitive to extremes) and population distributions (very heavy tails) are not well suited for this.