STAT 540
Class meeting 22
Monday, March 31, 2014

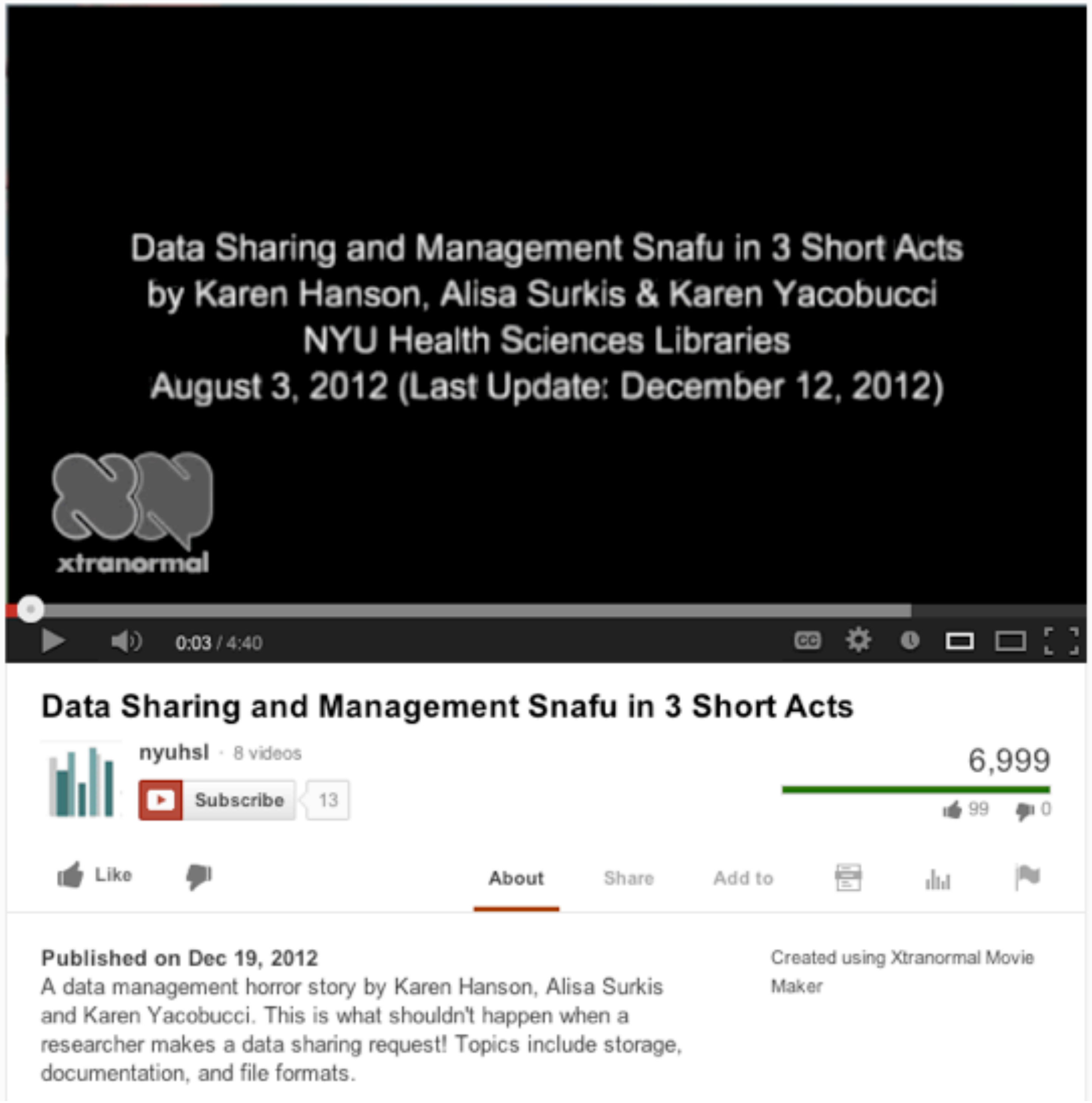
Dr. Jennifer (Jenny) Bryan
Department of Statistics and Michael Smith Laboratories

Resampling methods

first ~2/3 of class time was spent on bootstrap material; we covered up through using bootstrap to study the sampling distribution of the sample mean; rest was left for students to look at on their own

last ~1/3 of class time was spent discussing scientific workflow, esp. w.r.t. to STAT 540's coverage and usage of that; R, RStudio, R markdown, knitr, Git, GitHub, etc.
  - next two slides document that stuff, then bootstrap material follows

we watched a video about the importance of reproducible research, data and analytical documentation, data sharing


Data Sharing and Management Snafu in 3 Short Acts
by Karen Hanson, Alisa Surkis & Karen Yacobucci
NYU Health Sciences Libraries
August 3, 2012 (Last Update: December 12, 2012)

Data Sharing and Management Snafu in 3 Short Acts

Published on Dec 19, 2012
A data management horror story by Karen Hanson, Alisa Surkis and Karen Yacobucci. This is what shouldn't happen when a researcher makes a data sharing request! Topics include storage, documentation, and file formats.

Created using Xtranormal Movie Maker

https://www.youtube.com/watch?v=N2zK3sAtr-4&feature=youtu.be

linked to recent PLoS Comp Bio paper "Ten Simple Rules for Effective Computational Research" DOI: 10.1371/journal.pcbi.1003506 link

workarounds re: working directory for knitting .rmd and .r files when there is hierarchical directory structure
  - Project-level .RProfile, specific to each individual, in it assign the Project "home" directory to an R object, do NOT commit these .RProfile files to the shared Git repo
  - then use library(knitr); opts_knit$set(root.dir = "theProjDir",) in .rmd and .r files to finesse the working directory issue for all

important to do .rmd --> .md + figures and commit the downstream products when asking collaborators to view work ... be realistic about "activation energy" required to give feedback

the value of code review (not done well or at all in STAT 540 this year ...)

What I mean by "resampling methods":

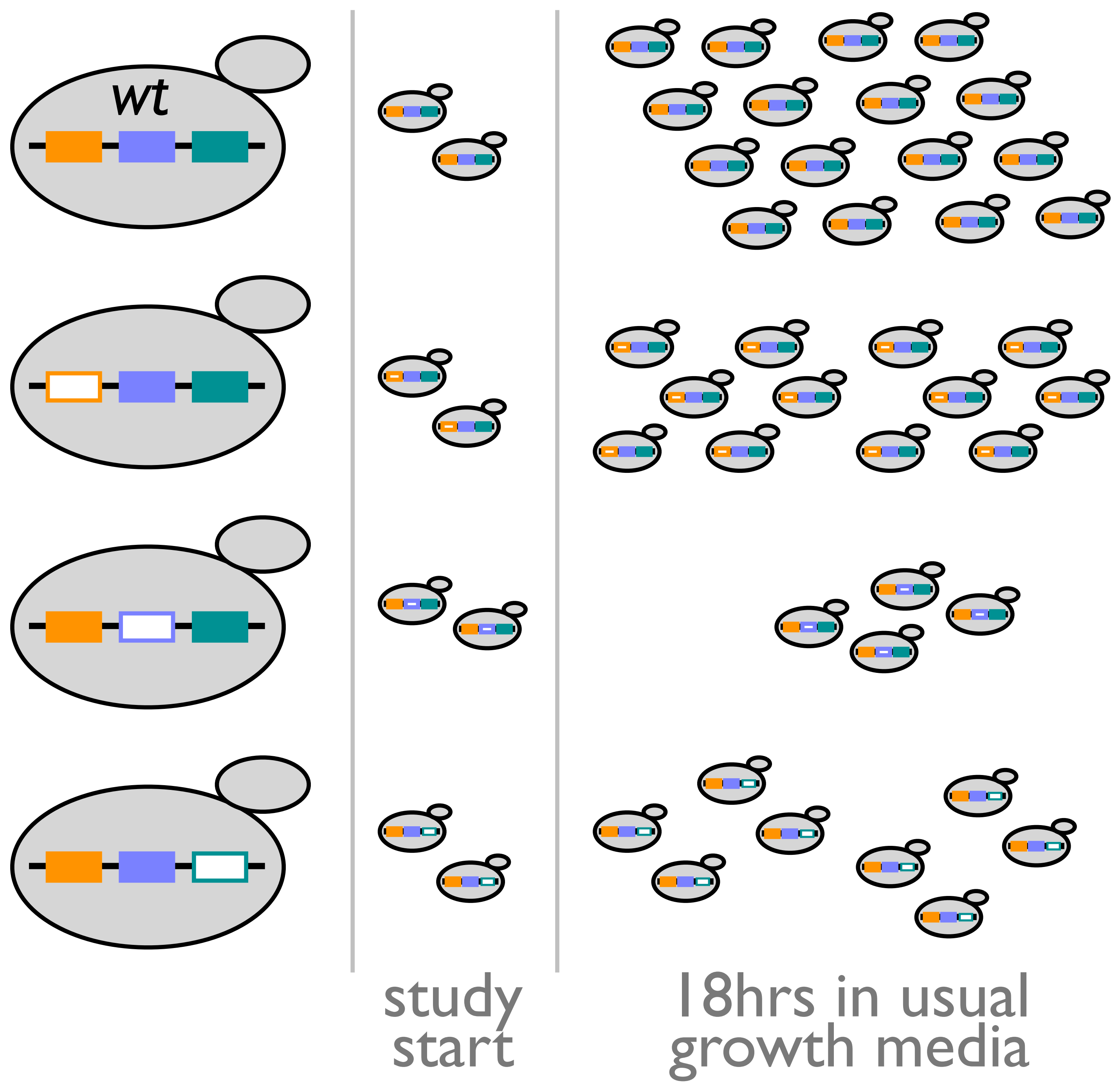
Ways of performing statistical inference that are *internal to the data*, i.e. you get the necessary knowledge about sampling variability from the observed data itself

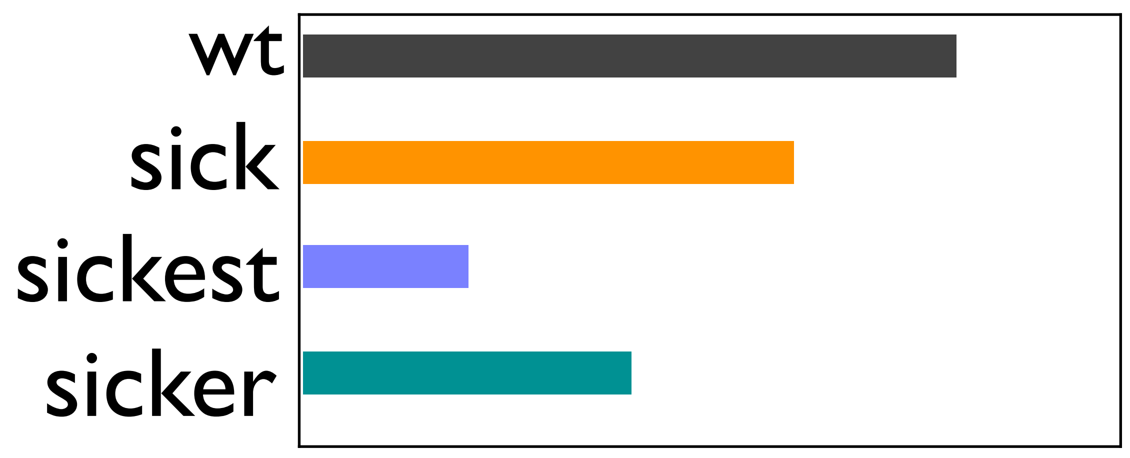
You pull yourself up by your bootstraps!

Keywords: resampling, bootstrap, permutation

Key points:
- substitute brute force computing for hard -- possibly intractable -- math
- reduce the need to make la-la land assumptions

Study of fitness of yeast deletion mutants





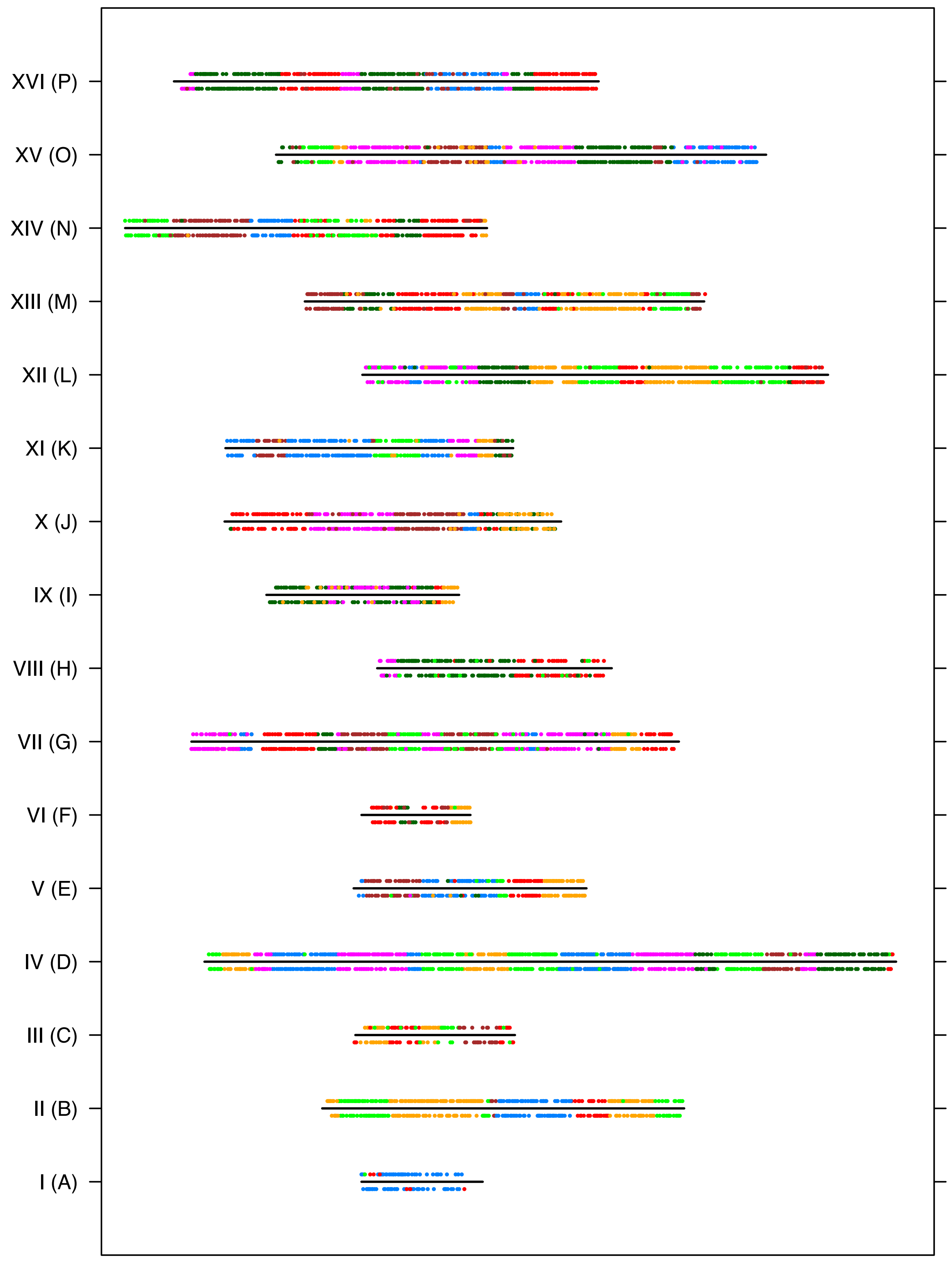
Rationale for growth studies of yeast deletion mutants

- Analogy: flipping circuit breakers in a house to determine which lights and outlets are controlled by each circuit
- If the deletion mutant for gene *g* is defective at some biological activity, that suggests that gene *g* contributes to that activity.
- Growth studies are the 'entry-level' study. In real life, we often measure more complicated phenotypes and subject the mutant to additional challenges, e.g. treatment with drugs or deletion/mutation of additional genes. Also, this type of data is often integrated with from other types of studies.

Yeast genome has 16 chromosomes.

Each gene lives somewhere on one of these chromosomes.

Therefore, each deletion mutant is also associated with one yeast chromosome.



response = a quantitative measure of growth

e.g. growth rate or # cells at study end

also know the specific yeast gene that was deleted

e.g. YDL133WY = a yeast ORF

and the chromosome on which the gene is found

e.g. "chromosome 4 / D"

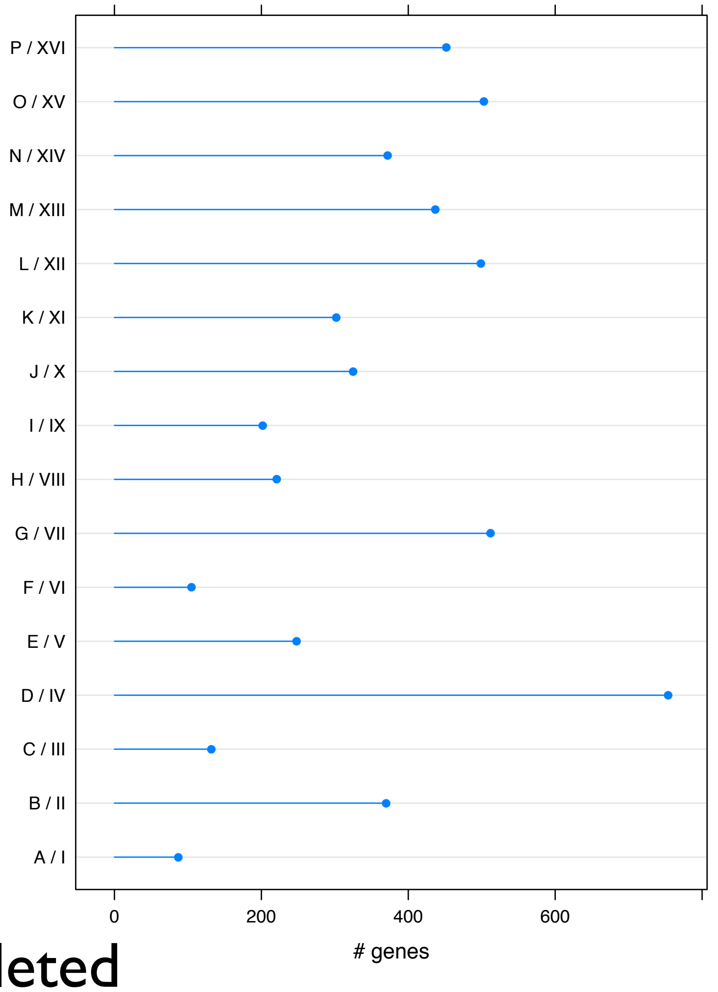
Minimum you need to know

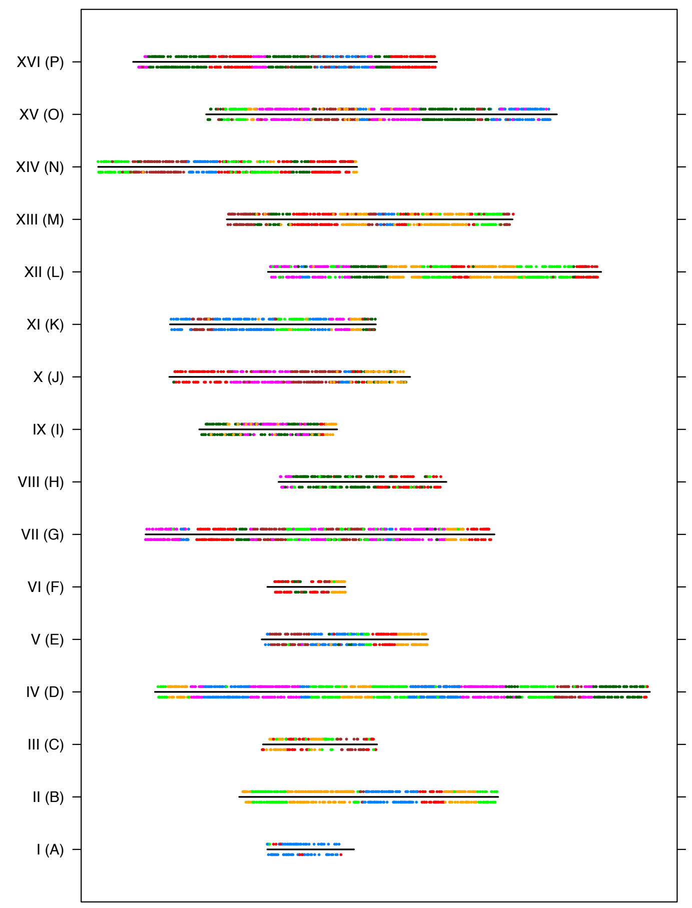
- Measuring growth -- an example of a phenotype -- on thousands of different strains of yeast. This is the response.
- Each strain is characterized by lacking the DNA for one gene.
- Each gene is found on one of the sixteen chromosomes in the yeast genome. This is the (potential) explanatory variable.
- It is interesting, for biological and experimental reasons, to ask whether the distribution of 'growth after gene deletion' differs across the chromosomes.

```
> str(hDat)
'data.frame':  5521 obs. of  4 variables:
 $ geneDel     : Factor w/ 5521 levels "YAL001C","YAL002W",..: 1 2 3 4 5 6 7 8..
 $ chromo      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ chromoPretty: Factor w/ 16 levels "A / I","B / II",..: 1 1 1 1 1 1 1 1 1 1 ..
 $ pheno       : num  9.39 9.4 10.38 10.54 8.65 ...
```

```
> peek(hDat)
     geneDel chromo chromoPretty    pheno
190  YBL102W      2        B / II 9.285750
917  YDR089W      4        D / IV 9.528659
1040 YDR185C      4        D / IV 7.079669
1969 YGL201C      7       G / VII 9.754082
2118 YGR046W      7       G / VII 9.262812
3175 YKL085W     11        K / XI 9.479903
3622 YLR176C     12       L / XII 7.359638
```

```
> dotplot(table(hDat$chromoPretty),
+         origin = 0, type = c("p", "h"),
+         xlab = "# genes")
```



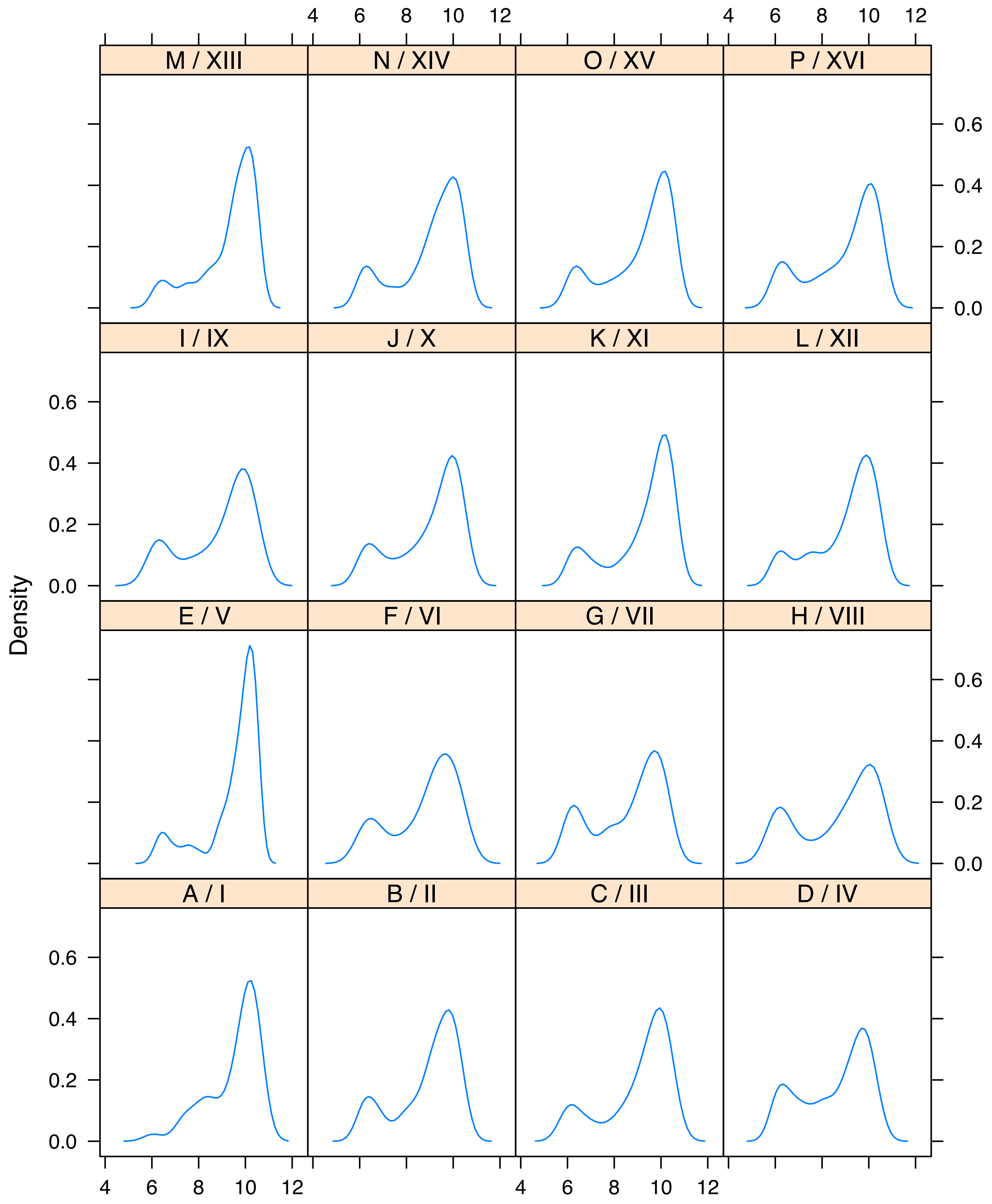


Each row consists of

- geneDel = name of the gene that was deleted
- chromo = the associated chromosome (an integer between 1 and 16)
- chromoPretty = a prettier version of the chromosome (more suitable for labeling in tables and figures)
- pheno = a growth phenotype (due to experimental realities and pre-processing, the units are meaningless, i.e. don't expect to see a cell count here)

quantitative growth phenotypes for gene deletion mutants

each panel = phenotypes for mutants lacking genes on that chromosome



Data source: Giaever G, Flaherty P, Kumm J, Proctor M, Nislow C, et al. (2004) Chemogenomic profiling: identifying the functional interactions of small molecules in yeast. Proc Natl Acad Sci U S A 101: 793-798. Pubmed. DOI: 10.1073/pnas.0307490100

All of my code, data, figures, and results for this example:
http://www.stat.ubc.ca/~jenny/notOcto/STAT545A/examples/yeastDeletionGrowth/

I have done some preprocessing and simplification of a subset of the data associated with the reference above. Specifically, we are looking at growth data on yeast deletion mutants growing in control conditions. Underlying dataset available supplementary information from the article above.

Typical applications of bootstrap

- Estimating key features of a sampling distribution
  - the standard deviation of a statistic (“standard error”)
  - the bias of an estimate
  - assess whether the asymptotic distribution has started to ‘kick in’ at a finite sample size
  - we’ll use bootstrap to look at the sample median
- Other inferential activities
  - Hypothesis testing (we’ll use bootstrap to conduct a one-sample test about the median and to revisit two-group comparison)
  - Confidence intervals

“Plug-in principle”

- Often we want to know something about the data-generating distribution F
- Specifically, we’re interested in a parameter $\theta = t(F)$
  - Example: expectation $\theta = t(F) = E_F(X)$
- What we have: a random sample from F
- Use it to get the empirical distribution function $\hat{F}$
- Estimate $t(F)$ with $t(\hat{F})$
  - Example cont’d: $\hat{\theta} = t(\hat{F}) = \frac{1}{n}\sum_i x_i$
- The bootstrap is an application of this “plug-in principle”

Bootstrap = Statistical Analogy

Here's an analogy --

HAND : PALM : : FOOT : _____

Bootstrap = Statistical Analogy

Here's an analogy --

HAND : PALM : : FOOT : SOLE

Here's an another --

$$\hat{\theta} : \theta :: \hat{\theta}^* : \hat{\theta}$$

Or, in English:

estimate : true parameter : : bootstrap estimate : estimate

Bootstrap = Statistical Analogy

$$\hat{\theta} : \theta :: \hat{\theta}^* : \hat{\theta}$$

estimate : true parameter : : bootstrap estimate : estimate

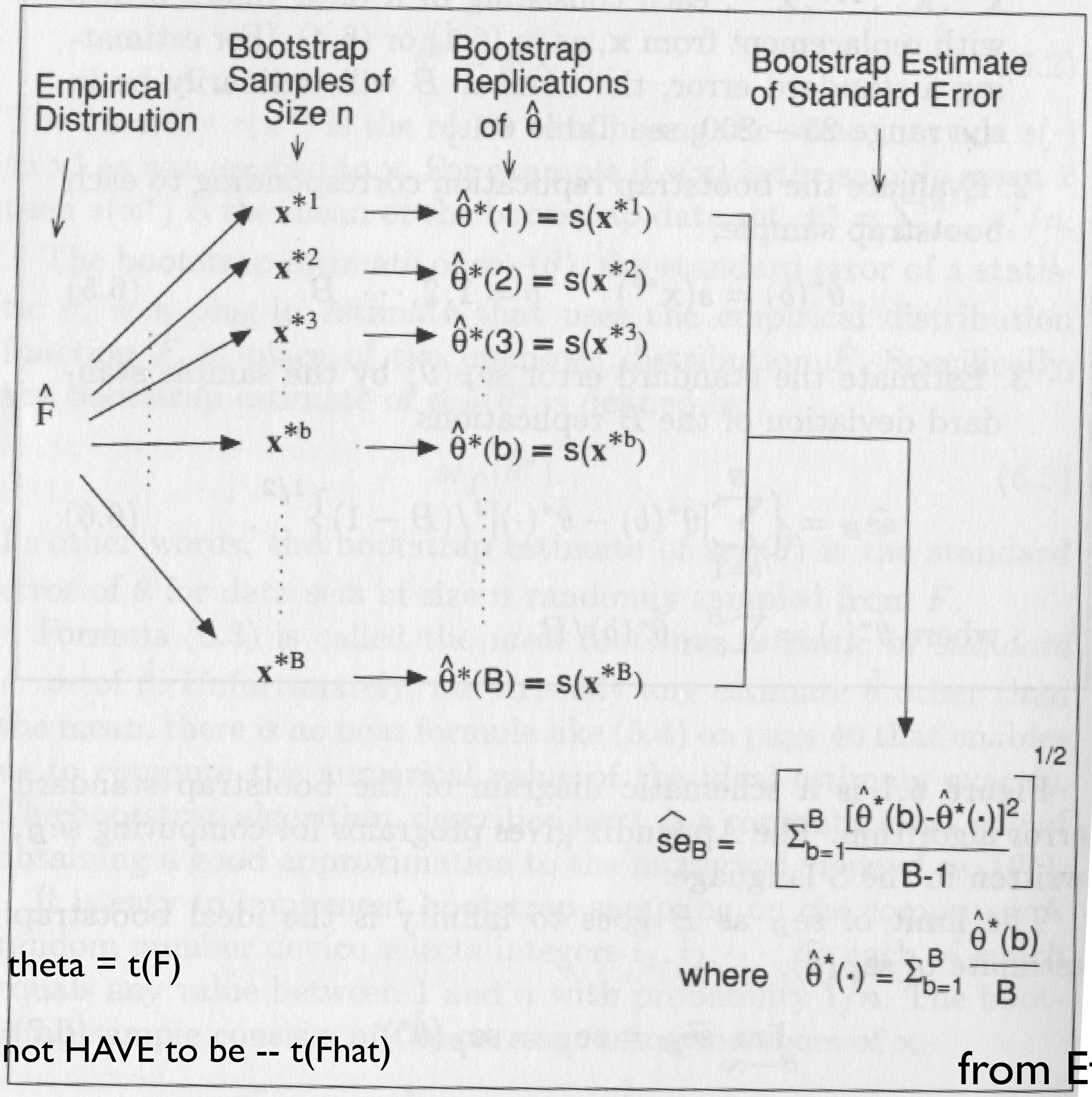
$\hat{\theta}^*$

Assess how $\hat{\theta}$ varies around θ by seeing how $\hat{\theta}^*$ varies around $\hat{\theta}$.

Bootstrap applied to sampling dist’n

- Imagine a simple ‘one-sample’ problem, simple one-dimensional real-valued parameter t(F) = θ
- We can use the distribution of bootstrap statistics to gain insight into
  - an estimator’s sampling distribution (Common question: Is it starting to look normal?)
  - the variance (or standard deviation) of the sampling dist’n (Addresses standard error)
  - the expectation (Addresses bias of the estimator)

Figure 6.1. *The bootstrap algorithm for estimating the standard error of a statistic $\hat{\theta} = s(\mathbf{x})$; each bootstrap sample is an independent random sample of size n from $\hat{F}$. The number of bootstrap replications B for estimating a standard error is usually between 25 and 200. As $B \to \infty$, $\widehat{se}_B$ approaches the plug-in estimate of $se_F(\hat{\theta})$.*


Empirical Distribution
Bootstrap Samples of Size n
Bootstrap Replications of $\hat{\theta}$
Bootstrap Estimate of Standard Error
$\hat{F}$
$\mathbf{x}^{*1}$
$\mathbf{x}^{*2}$
$\mathbf{x}^{*3}$
$\mathbf{x}^{*b}$
$\mathbf{x}^{*B}$
$\hat{\theta}^*(1) = s(\mathbf{x}^{*1})$
$\hat{\theta}^*(2) = s(\mathbf{x}^{*2})$
$\hat{\theta}^*(3) = s(\mathbf{x}^{*3})$
$\hat{\theta}^*(b) = s(\mathbf{x}^{*b})$
$\hat{\theta}^*(B) = s(\mathbf{x}^{*B})$
$\widehat{se}_B = \left[\sum_{b=1}^{B} \frac{[\hat{\theta}^*(b) - \hat{\theta}^*(\cdot)]^2}{B-1}\right]^{1/2}$
where $\hat{\theta}^*(\cdot) = \sum_{b=1}^{B} \frac{\hat{\theta}^*(b)}{B}$

Notation:

s(x) is an estimator for theta = t(F)

It might be -- but does not HAVE to be -- t(Fhat)

from Efron & Tibshirani

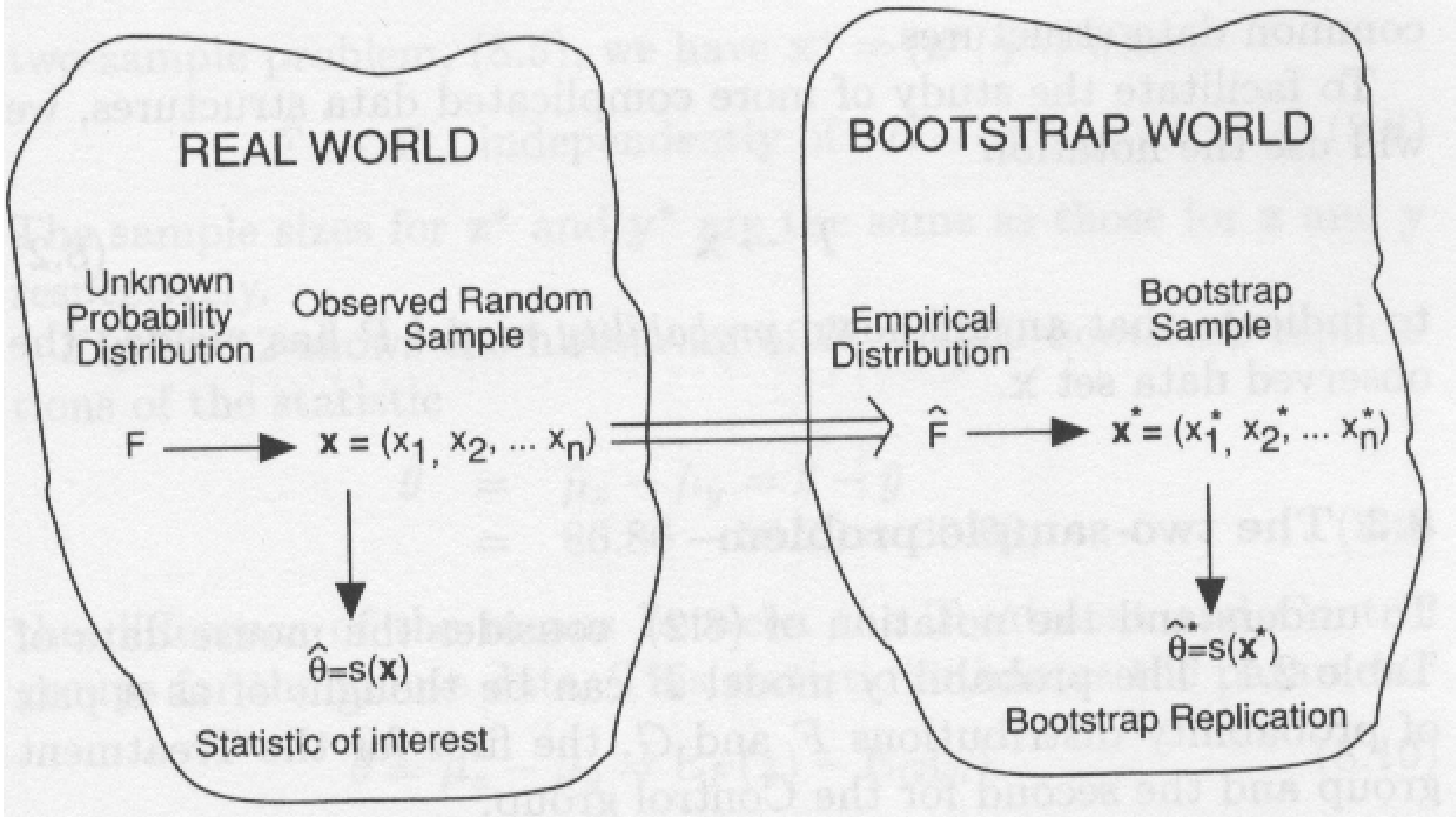
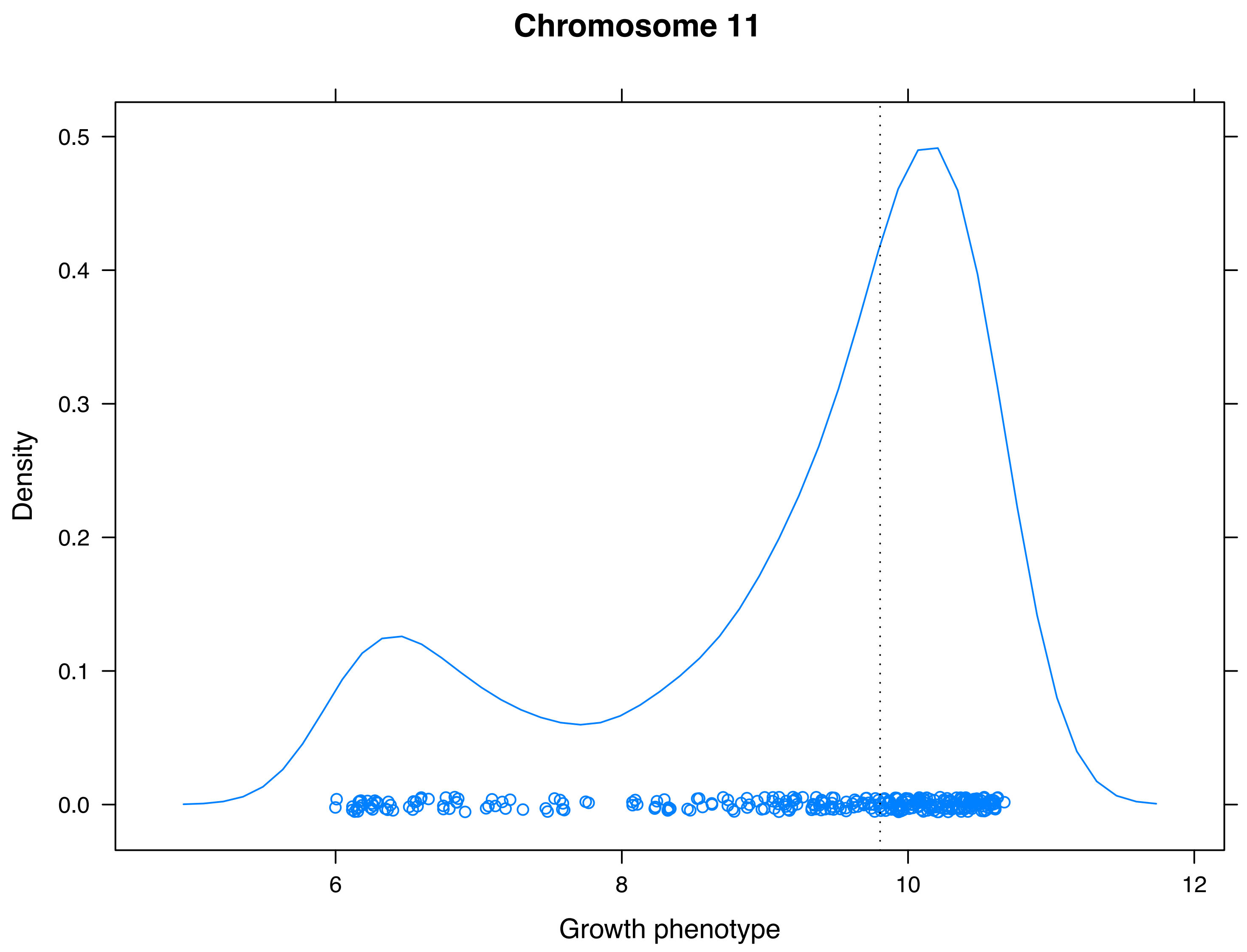


Figure 8.1. *A schematic diagram of the bootstrap as it applies to one-sample problems. In the real world, the unknown probability distribution* F *gives the data* $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ *by random sampling; from* $\mathbf{x}$ *we calculate the statistic of interest* $\hat{\theta} = s(\mathbf{x})$. *In the bootstrap world,* $\hat{F}$ *generates* $\mathbf{x}^*$ *by random sampling, giving* $\hat{\theta}^* = s(\mathbf{x}^*)$. *There is only one observed value of* $\hat{\theta}$, *but we can generate as many bootstrap replications* $\hat{\theta}^*$ *as affordable. The crucial step in the bootstrap process is "*$\Longrightarrow$*", the process by which we construct from* $\mathbf{x}$ *an estimate* $\hat{F}$ *of the unknown population* F.

from Efron & Tibshirani

Example: Median for chromosome 11

```
> jChromo <- 11
> x <- hDat$pheno[hDat$chromo == jChromo]

> (nx <- length(x))
[1] 302

> (jMedian <- median(x))
[1] 9.804809
```



Example: Median of phenotypes for chromosome 11

- Large-sample theory says the sample median is asymptotically normal with mean = true median = *m* and variance 1 / 4n f(*m*)^2
- Good news = asymptotic dist'n is known
- Bad news = depends on the true density at the median*
- Let's compare this theoretical, asymptotic result to the bootstrap result.

*If I knew the density, I'd know the median, wouldn't I?

Literally:
I’m going to draw tons of new samples from the empirical distribution. These are known as “bootstrap samples”.

I’m going to compute the median for each. You might call these “bootstrap statistics”.

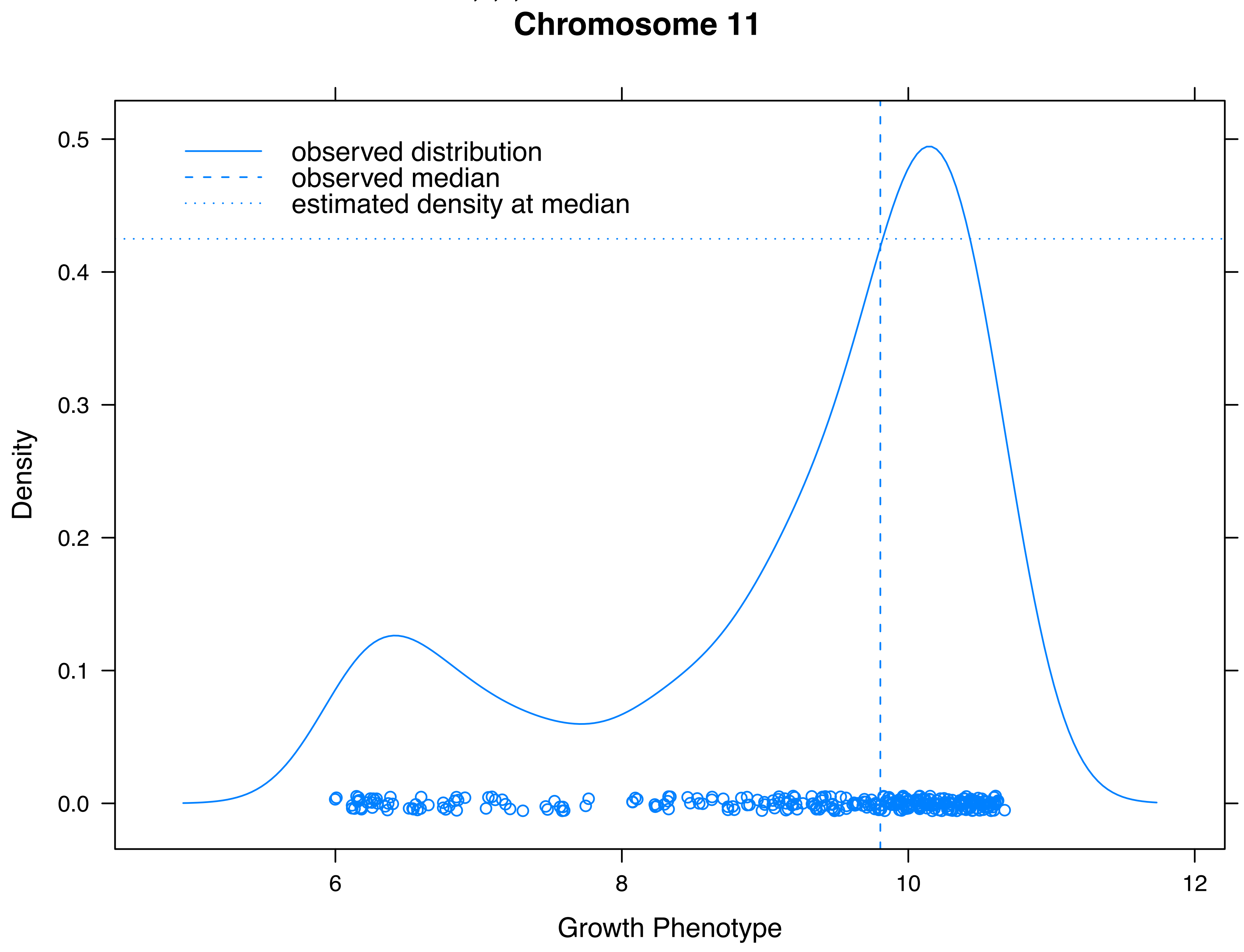
I’m going to show you their empirical distribution -- call that the “bootstrap sampling distribution” -- and superpose the theoretical normal sampling distribution.

We’re going to see how close they are.

```
> ## preparing to do inference about the median
> estDens <- density(x, n = 200)

> class(estDens)
[1] "density"

> names(estDens)
[1] "x"         "y"         "bw"        "n"         "call"      "data.name"
[7] "has.na"

> (fHatAtMedian <- estDens$y[which.min(abs(estDens$x - jMedian))])
[1] 0.4249459

> (theorStdErr <- sqrt(1 / (4 * nx * fHatAtMedian^2)))
[1] 0.0677069
```

- Recall, large-sample theory says the sample median is asymptotically normal with mean = true median = m and variance $1 / 4n\, f(m)^2$

```
> (jMedian <- median(x))                        # 9.80 for chr11
[1] 9.804809

...

> (fHatAtMedian <- estDens$y[which.min(abs(estDens$x - jMedian))])
[1] 0.4249459
> (theorStdErr <- sqrt(1 / (4 * nx * fHatAtMedian^2)))
[1] 0.0677069
```



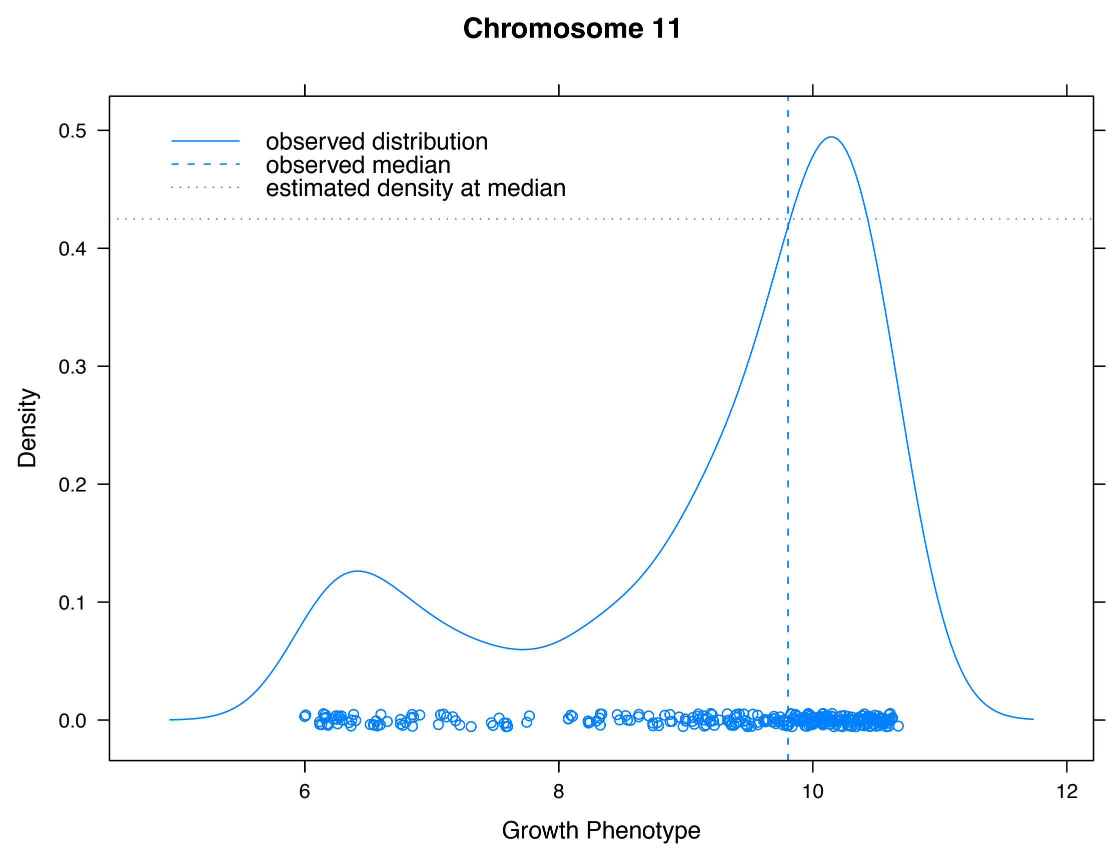
now I draw a bootstrap sample:

literally I draw a new sample of size n = 302 from the observed data, with replacement

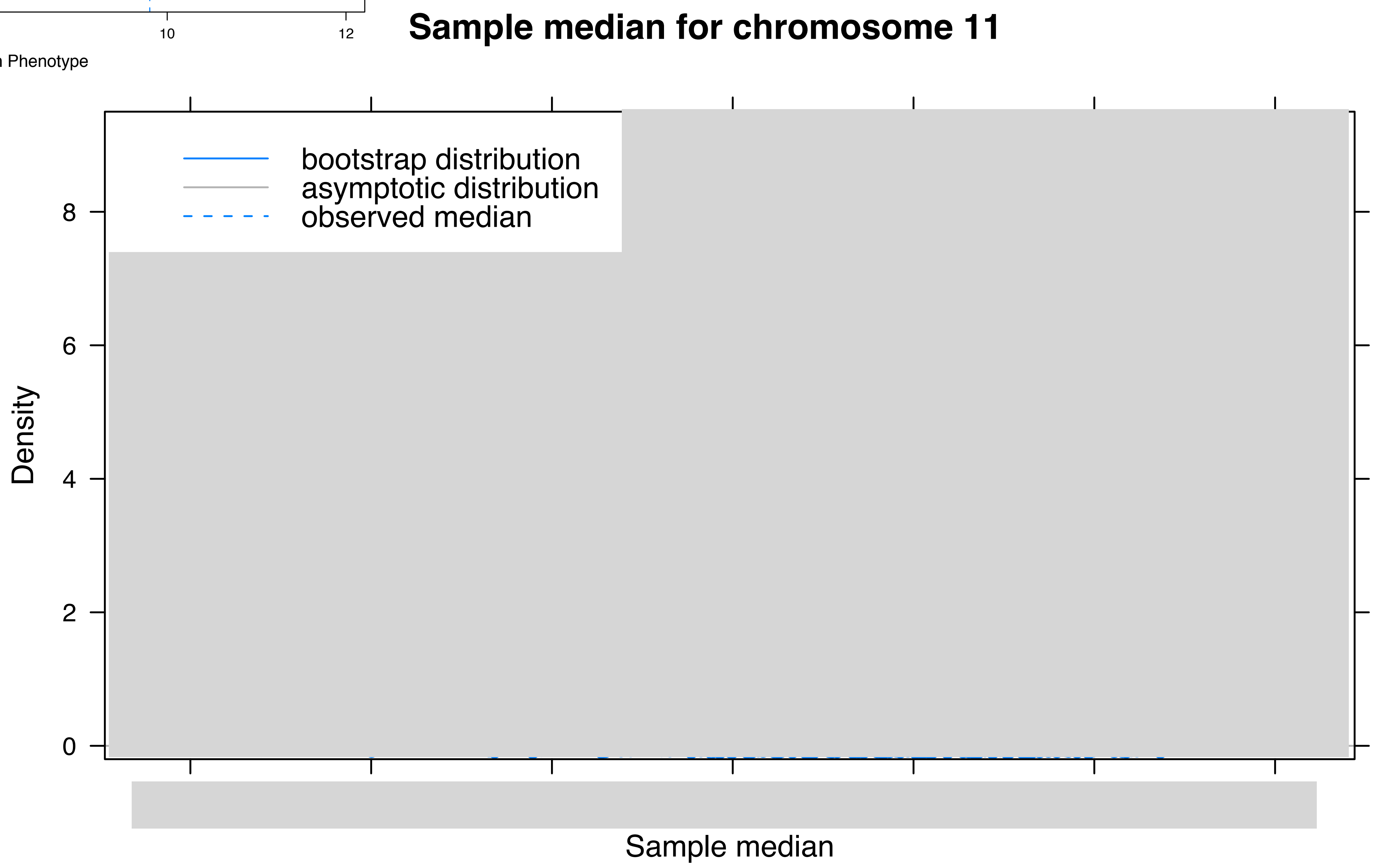
some observations will re-appear ... some once, some twice, etc. .... some don't show up in the bootstrap sample at all

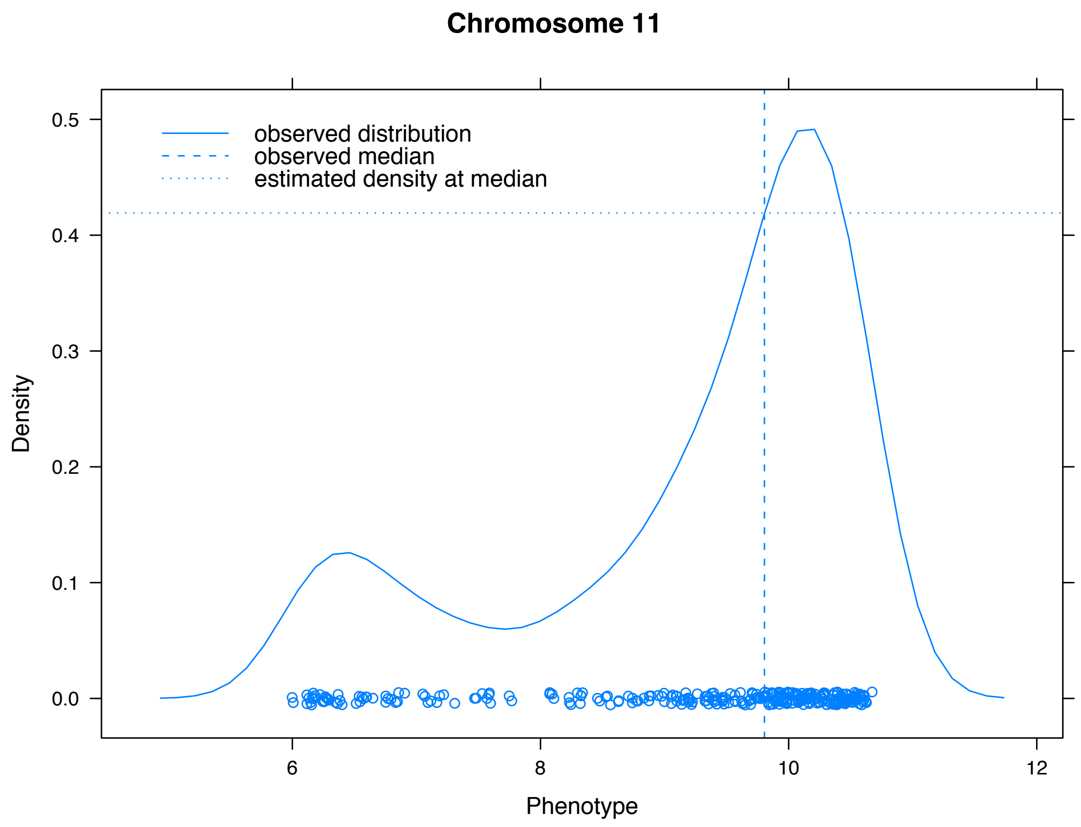
I take the median of the bootstrap sample. That is a bootstrap statistic.

I do that B times. B is a big number.



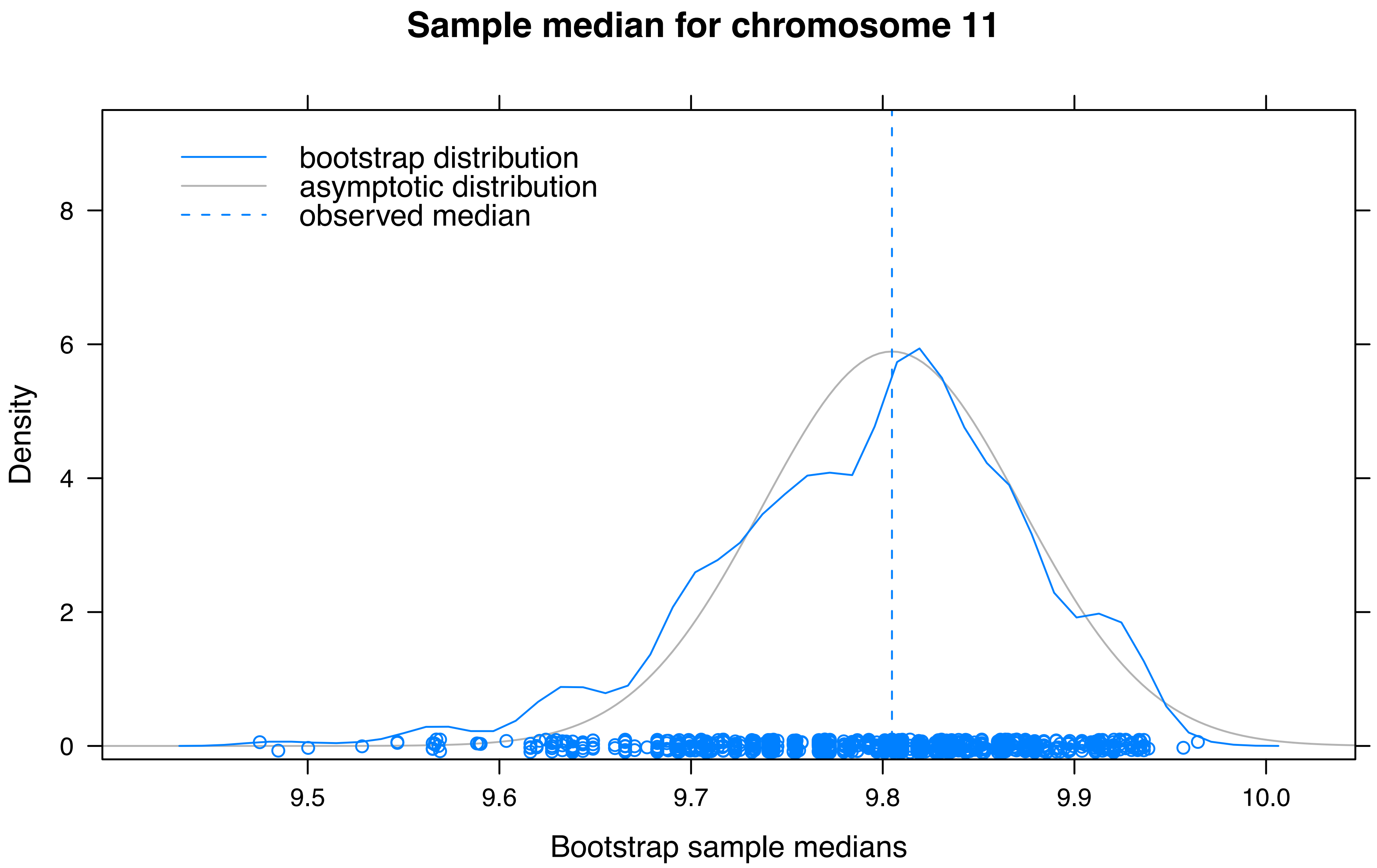
Can you fill in the grey areas?
Give it a try!

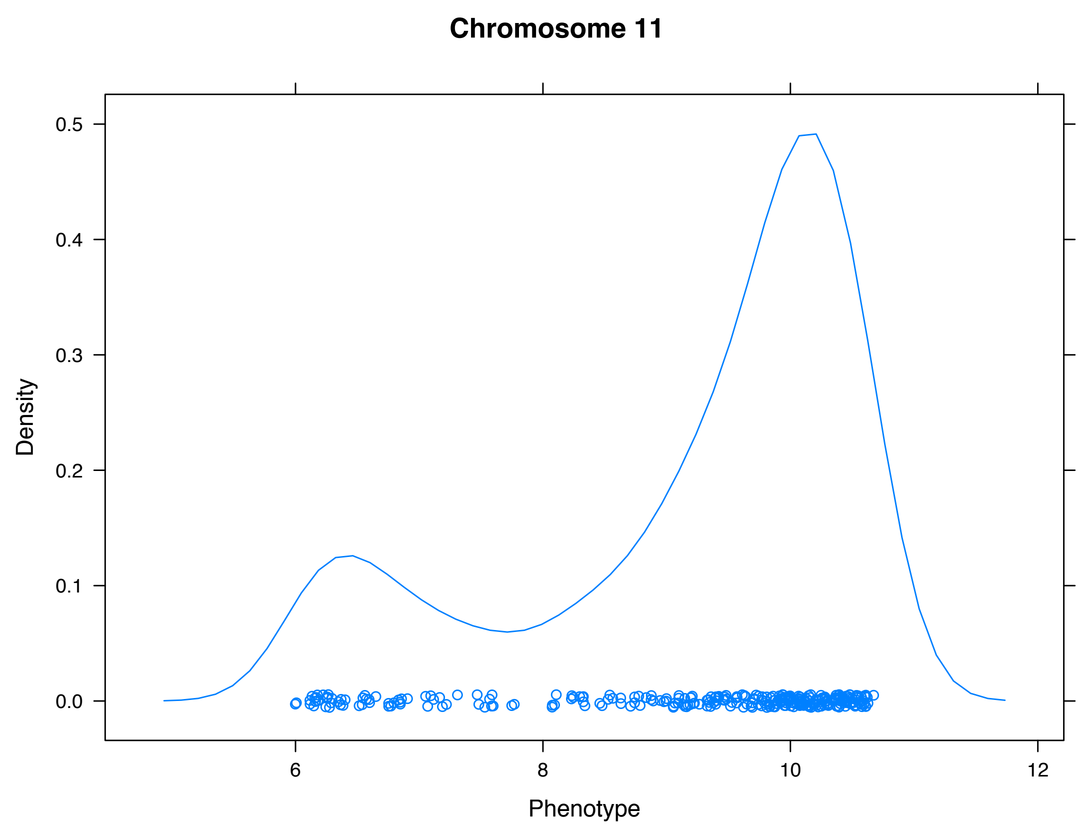


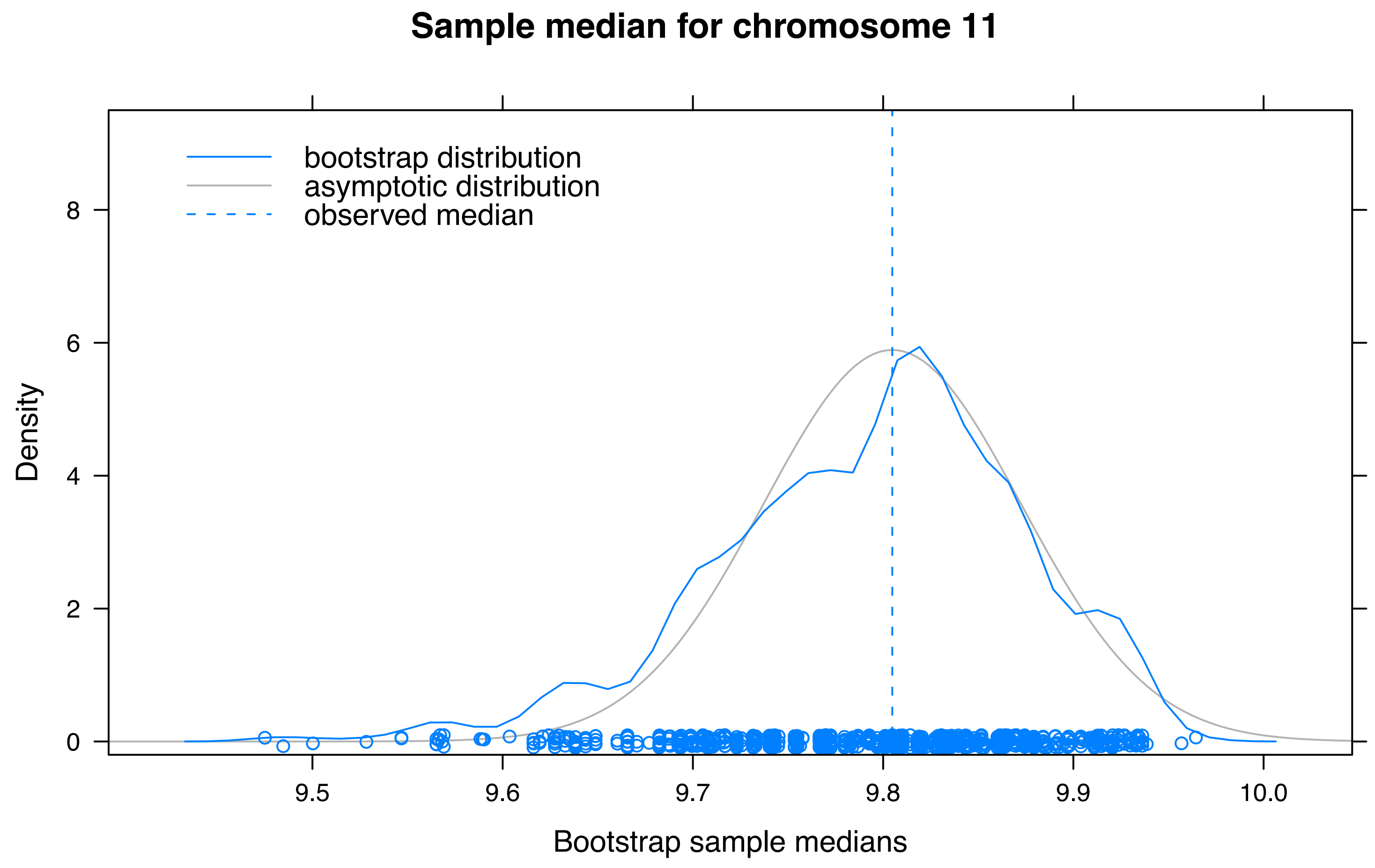


Features we could foresee:

- Both dist'ns have mode @ sample median = 9.8
- Left tail of bootstrap distribution heavier than than that of asymp. norm







```
> B <- 1000

> bootData <-
+   matrix(sample(x, size = B * nx, replace = TRUE),
+          nrow = nx, ncol = B)

> bootTestStat <- apply(bootData, 2, median)

> (bootStdErr <- sqrt(var(bootTestStat)))
[1] 0.07937564

> theorStdErr
[1] 0.0677069

> mean(bootTestStat)
[1] 9.796118

> jMedian
[1] 9.804809

> abs(mean(bootTestStat) - jMedian)/bootStdErr
[1] 0.1094916
```

I conclude ... for a data-generating distribution as bimodal as this, n = 300 is close to -- but not quite in -- Asymptopia.

Good default template for conducting a bootstrap. Can be adapted for other resampling or random data generation tasks.

```
> B <- 1000
```

make it easy to start w/ small B, then scale up

```
> bootData <-
+   matrix(sample(x, size = B * nx, replace = TRUE),
+          nrow = nx, ncol = B)
```

generate the bootstrap data all at once

```
> bootTestStat <- apply(bootData, 2, median)
```

use data aggregation techniques to compute bootstrap statistics

```
> (bootStdErr <- sqrt(var(bootTestStat)))
[1] 0.08163377

> mean(bootTestStat)
[1] 9.796118

> jMedian
[1] 9.804809

> abs(mean(bootTestStat) - jMedian)/bootStdErr
[1] 0.1094916
```

R packages for bootstrapping

- I don't really use these ... always seems easier to just do it myself.
  - boot, a companion to another book ("Bootstrap Methods and Their Applications" by A. C. Davison and D.V. Hinkley (1997, CUP)) -- seems to be distributed with R
  - bootstrap, companion to Efron and Tibshirani's book, seems not to be actively maintained, new work is encouraged to use boot

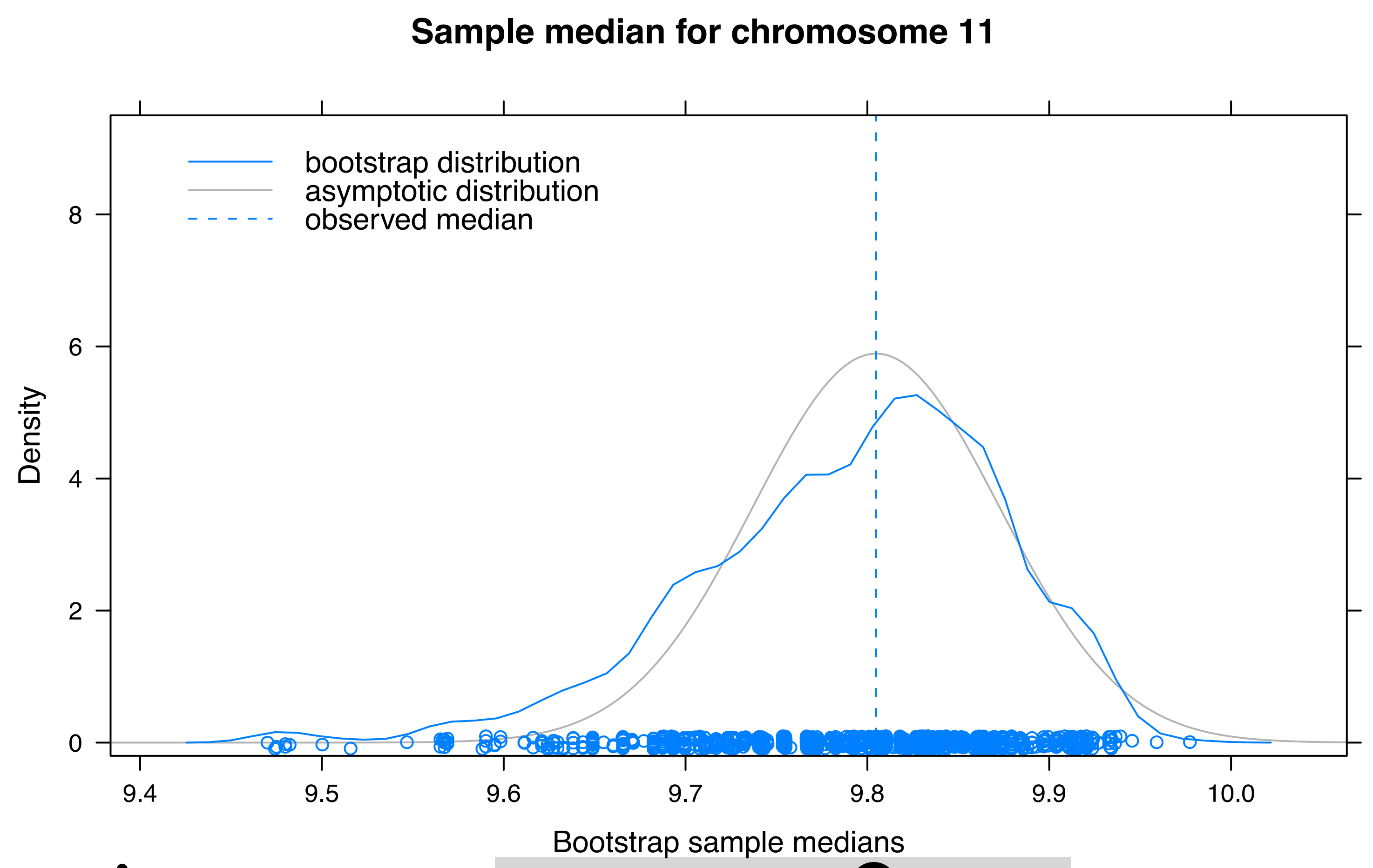
Using the boot package

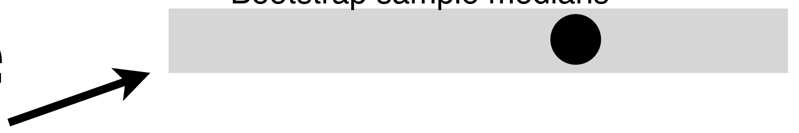
boot output

```
> bootRes <- boot(x, function(z, i) median(z[i]), R = 1000)

> bootRes

ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = x, statistic = function(z, i) median(z[i]), R = 1000)

Bootstrap Statistics :
    original       bias    std. error
t1* 9.804809 -0.01221307  0.08194345
```

results from my implementation

```
> jMedian
[1] 9.804809

> mean(bootTestStat) - jMedian
[1] -0.008690966

> bootStdErr
[1] 0.07937564
```





an interval estimate for the median

```
> boot.ci(bootRes, conf = c(0.90, 0.95), type = "all")
...
Intervals :
Level      Normal              Basic
90%   ( 9.682,  9.952 )   ( 9.692,  9.971 )
95%   ( 9.656,  9.978 )   ( 9.683, 10.019 )

Level     Percentile            BCa
90%   ( 9.638,  9.918 )   ( 9.621,  9.913 )
95%   ( 9.590,  9.927 )   ( 9.579,  9.921 )
```

Bootstrap methods can be used to build CIs. Here showing output from ‘boot’ package, boot.ci() function.

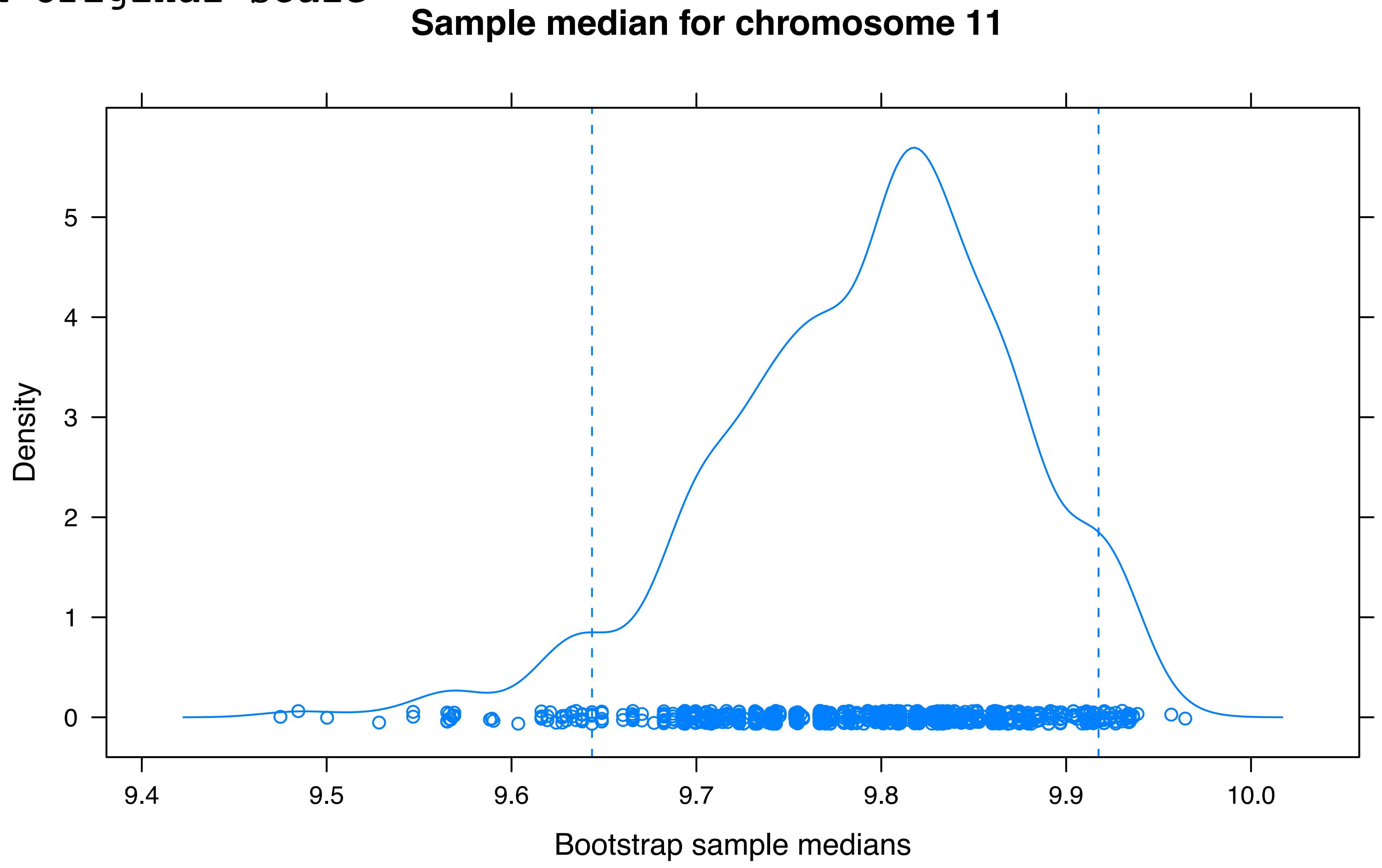
Percentile confidence interval

```
> confLevel <- 0.90

> alpha <- 1 - confLevel

> quantile(bootTestStat, probs = c(alpha / 2, 1 - alpha / 2))
      5%      95% 
9.638231 9.910952 

> boot.ci(bootRes, conf = confLevel, type = "perc")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL : 
boot.ci(boot.out = bootRes, conf = confLevel, type = "perc")

Intervals : 
Level     Percentile     
90%   ( 9.638,  9.918 )  
Calculations and Intervals on Original Scale
```

Easiest to understand, but not your best long-term general choice.

Percentile confidence interval

```
> quantile(bootTestStat, probs = c(alpha / 2, 1 - alpha / 2))
      5%      95%
9.638231 9.910952

> boot.ci(bootRes, conf = confLevel, type = "perc")
...
Intervals :
Level     Percentile
90%   ( 9.638,  9.918 )
Calculations and Intervals on Original Scale
```

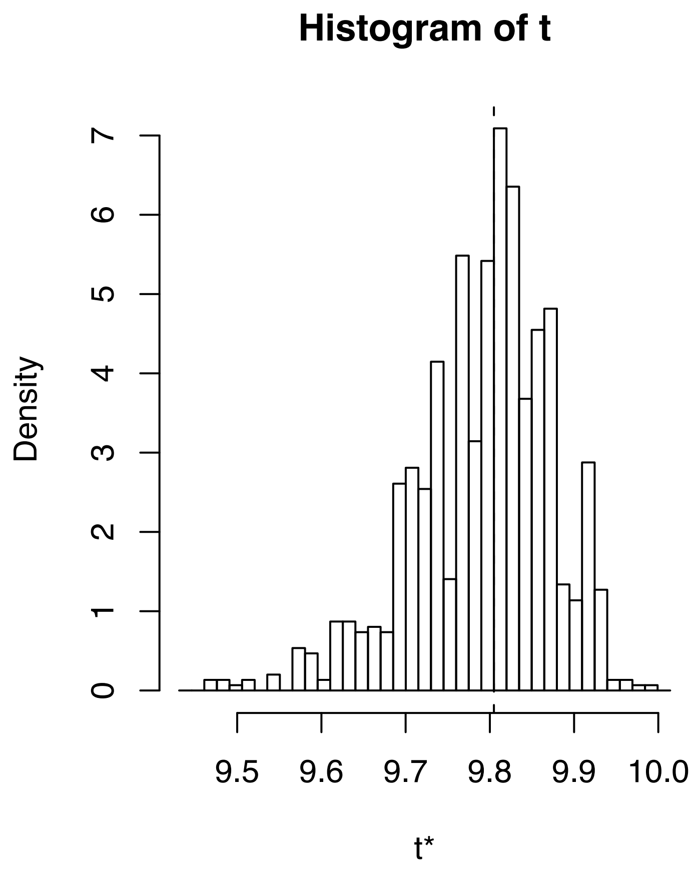


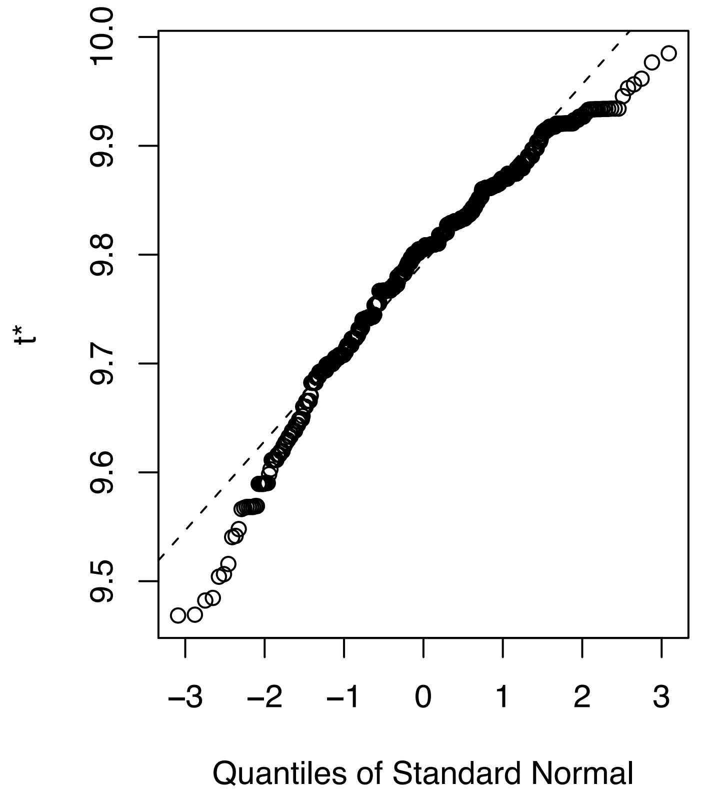
‘Basic’ confidence interval

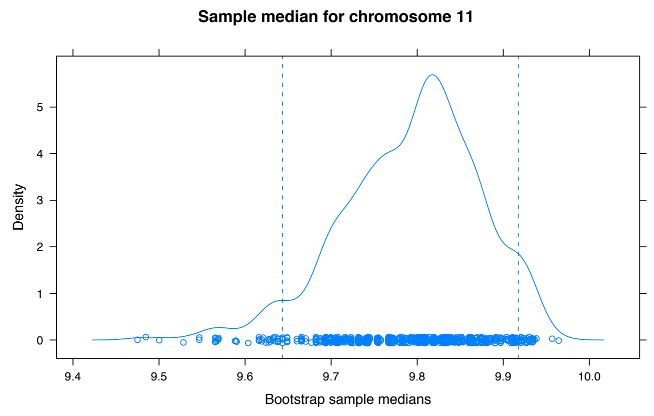
```
> ## 'basic' confidence interval
> ctrdBootMedians <- bootTestStat - jMedian

> (critVals <- quantile(ctrdBootMedians, probs = c(alpha / 2, 1 - alpha / 2)))
        5%        95% 
-0.1665784  0.1061428 

> c(jMedian - critVals[2], jMedian - critVals[1])
     95%       5% 
9.698666 9.971387 

> boot.ci(bootRes, conf = confLevel, type = "basic")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL : 
boot.ci(boot.out = bootRes, conf = confLevel, type = "basic")

Intervals : 
Level      Basic         
90%   ( 9.692,  9.971 )  
Calculations and Intervals on Original Scale
```

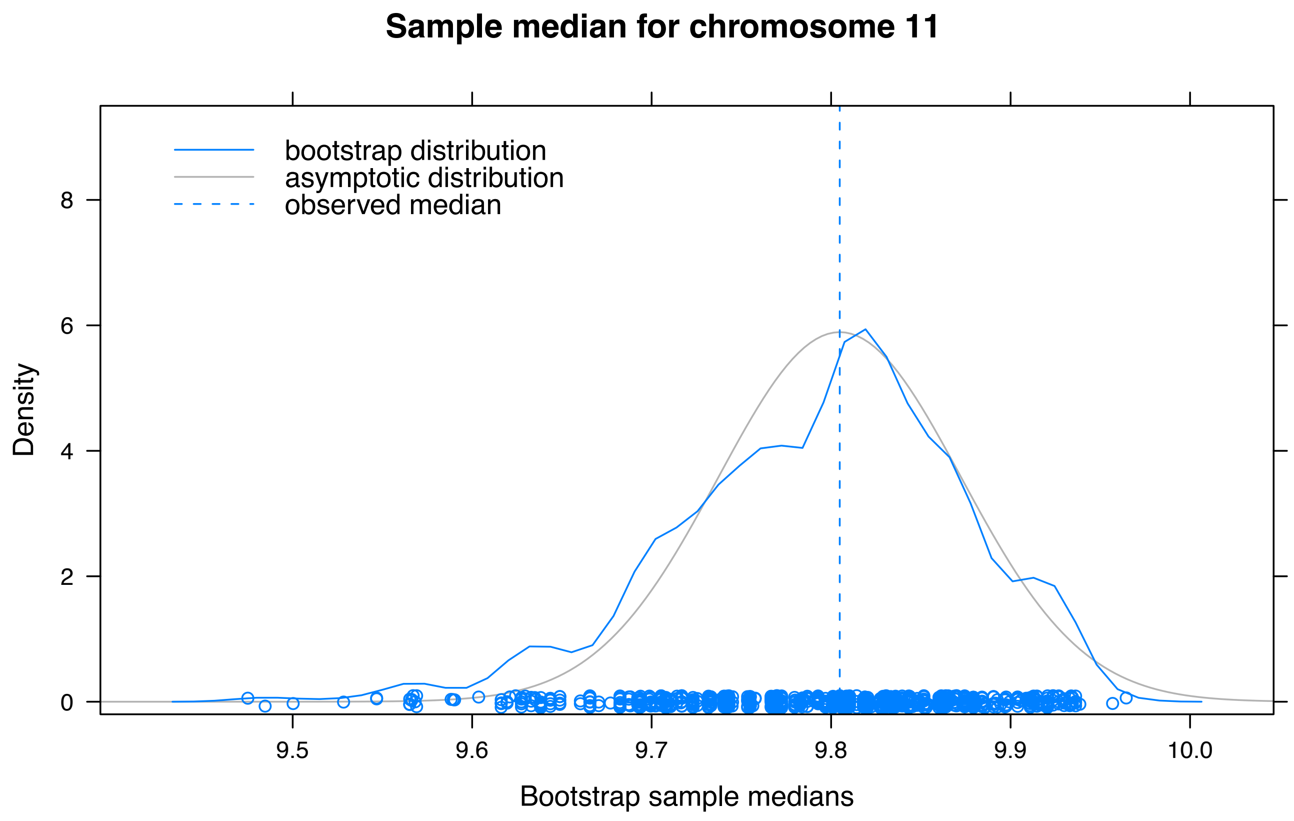
Performs better than percentile intervals, but less intuitive. Also consider BCa intervals.

> plot(bootRes)









JB's plots

How large should B be?

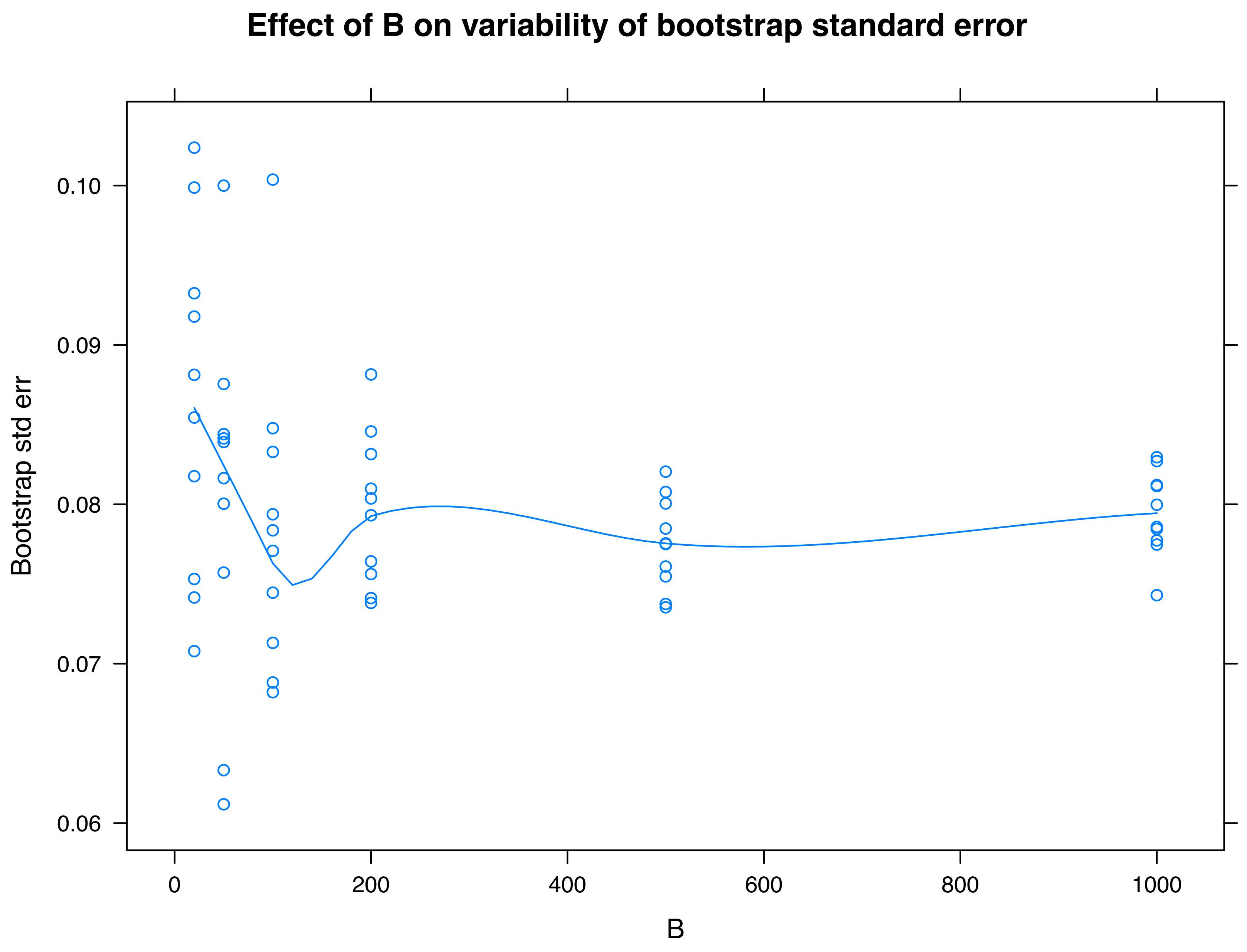
- Efron & Tibshirani seem very comfortable with B = 200 for the purposes of estimating standard error
- You will need more -- generally much much more -- for confidence intervals. Depends on method. Beyond our scope today.
- I often default to B = 1000 for std err estimation or testing, but frequently use much larger B, such as 10000. Why not?

```
> ## impact of B of estimation of standard error
> manyB <- c(20, 50, 100, 200, 500, 1000)

> ## I'm being a bit sloppy here, i.e. using x and nx, etc. from the
> ## environment; also hardwired for t(F) = median
> ## all of that would, ideally, be passed as arguments
> getBootstrapStdErr <- function(myB) {
+   bootData <-
+     matrix(sample(x, size = myB * nx, replace = TRUE),
+            nrow = nx, ncol = myB)
+   return(sqrt(var(apply(bootData, 2, median))))
+ }

> m <- 10                                  # reps per B
> system.time(
+ effectOfB <- sapply(manyB,
                     function(BB) replicate(m, getBootstrapStdErr(BB)))
+ +             )                           # never more than 2 sec for me
  user  system elapsed
 2.221   0.104   2.327
```

Getting 10 bootstrap estimates of standard error at various values of B. Note that this is all based on the *same observed dataset*. system.time() is good to know.

```
> colnames(effectOfB) <- paste("B",manyB, sep = "")
> head(effectOfB)
            B20        B50       B100       B200       B500      B1000
[1,] 0.07531774 0.08003954 0.07937572 0.08814994 0.07547978 0.07858142
[2,] 0.08176429 0.09999674 0.07130929 0.07641767 0.08205215 0.07997457
[3,] 0.07415397 0.08755235 0.08328948 0.07411757 0.08005469 0.08114402
[4,] 0.09324529 0.08413617 0.10037547 0.07931392 0.07609688 0.07429938
[5,] 0.10237819 0.06118017 0.08477801 0.07381404 0.07374786 0.08271338
[6,] 0.08544778 0.08391218 0.07708429 0.08036981 0.08077487 0.08295626
> effectOfB <- stack(as.data.frame(effectOfB))
> effectOfB$B <- rep(manyB, each = m)
> head(effectOfB)
      values ind  B
1 0.07531774 B20 20
2 0.08176429 B20 20
3 0.07415397 B20 20
4 0.09324529 B20 20
5 0.10237819 B20 20
6 0.08544778 B20 20
```

stack() is useful when repackaging ‘short and fat’ data into something ‘tall and skinny,’ which you will need to do often if you are a heavy user of tapply & friends and lattice.



```
xyplot(values ~ B, effectOfB,
       type = c('p','smooth'), ylab = "Bootstrap std err",
       main = "Effect of B on variability of bootstrap standard error")
```

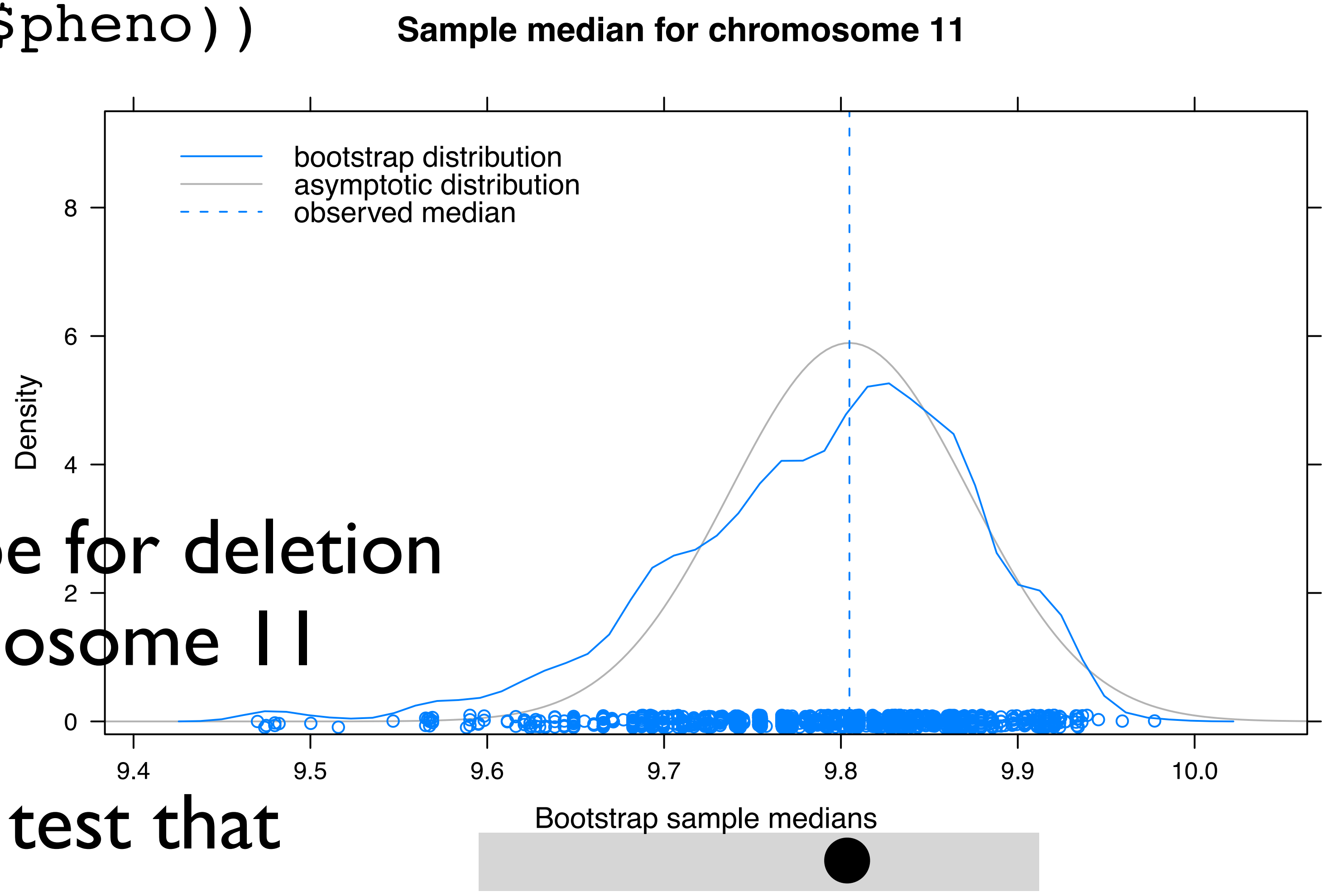
How large should B be?

- Previous figure makes B = 200 or, esp., B = 500 to 1000, seem quite reasonable for the purposes of estimating standard error.
- You will need more -- generally much much more -- for confidence intervals. Depends on method. Beyond our scope today.
- You see that it is not particularly burdensome to use fairly large B -- don't be timid. But ALSO don't write a loop!

Conducting a hypothesis test

- Overall median of phenotype = m_{all}
- Test whether chromosome-specific median = m_{all}
- Classical test, bootstrap test

```
> ## testing if the median = a particular value
> (medianNull <- median(hDat$pheno))
[1] 9.451124

> jMedian
[1] 9.804809
```



Is the median growth phenotype for deletion mutants associated with chromosome 11 equal to 9.45?
(No, of course it's not, but let's test that formally.)

One ‘classical’ way to test this

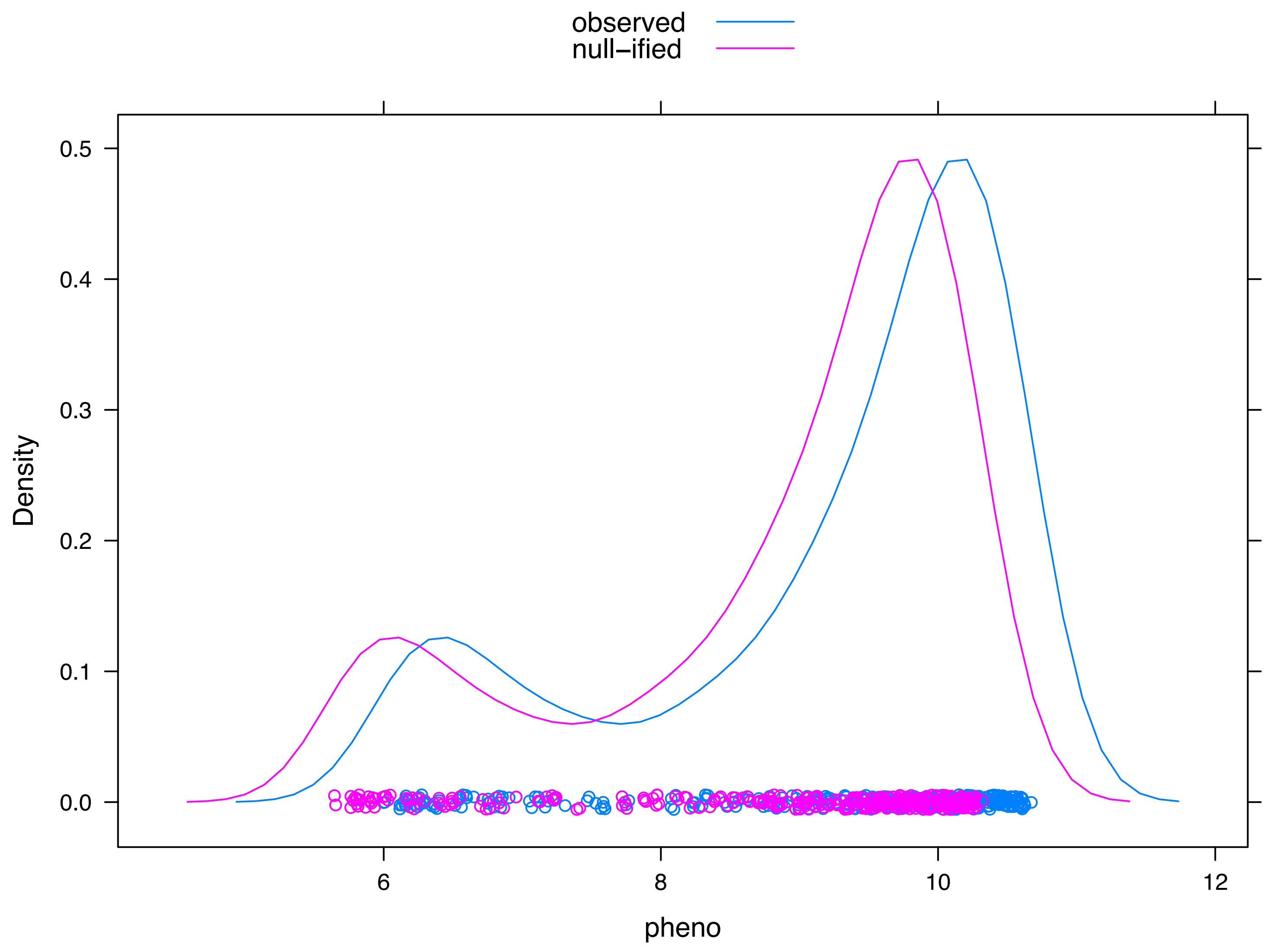
- If m_{all} is the true median, the probability of observed data being less than (or greater than) m_{all} equals 0.5.
- Under the null, the statistic = #x’s < m_{all} is Binomial(n = 302, p = 0.5).

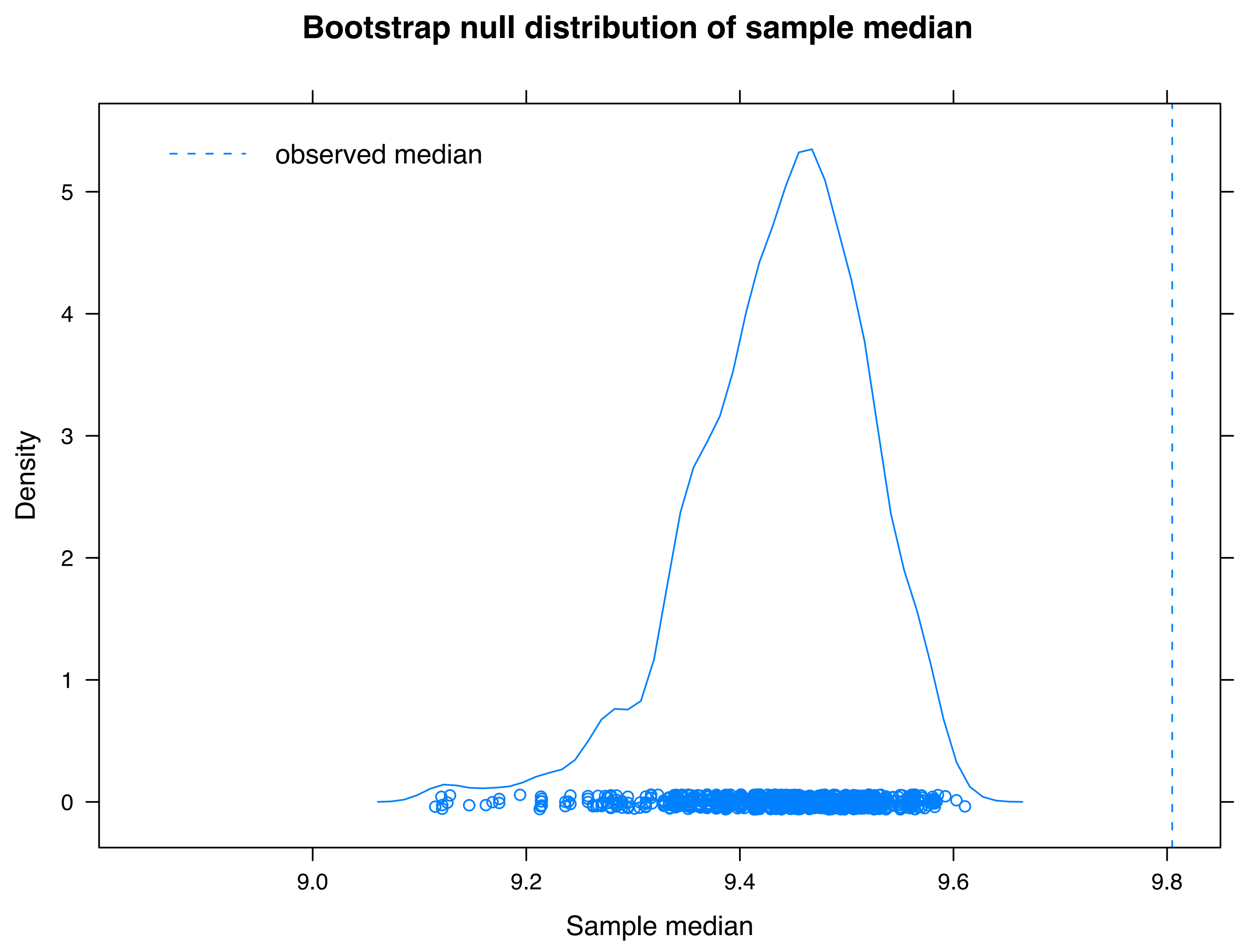
```
> binom.test(sum(x < medianNull), nx)

      Exact binomial test

data:  sum(x < medianNull) and nx
number of successes = 122, number of trials = 302, p-value = 0.001006
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.3481601 0.4616901
sample estimates:
probability of success
             0.4039735

> binom.test(sum(x >= medianNull), nx)

      Exact binomial test

data:  sum(x >= medianNull) and nx
number of successes = 180, number of trials = 302, p-value = 0.001006
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.5383099 0.6518399
sample estimates:
probability of success
             0.5960265
```

Bootstrap approach

- Perform the minimal transformation of the observed data to enforce the null hypothesis
  - "null-ified" data = obs. data - m_{chromo} + m_{all}
- Create bootstrap datasets from this "null-ified" version of the observed data
- Compute bootstrap test statistics
- Note the significance of the observed test statistic

```
> ## very explicit demo of "enforce the null"!
> xNull <- x - jMedian + medianNull
> foo <-
+   data.frame(pheno = c(x, xNull),
+              origin = factor(rep(c("observed","null-ified"),
+                each = nx),
+                levels = c("observed","null-ified")))
> densityplot(~ pheno, foo, groups = origin,
+             auto.key = TRUE)
```





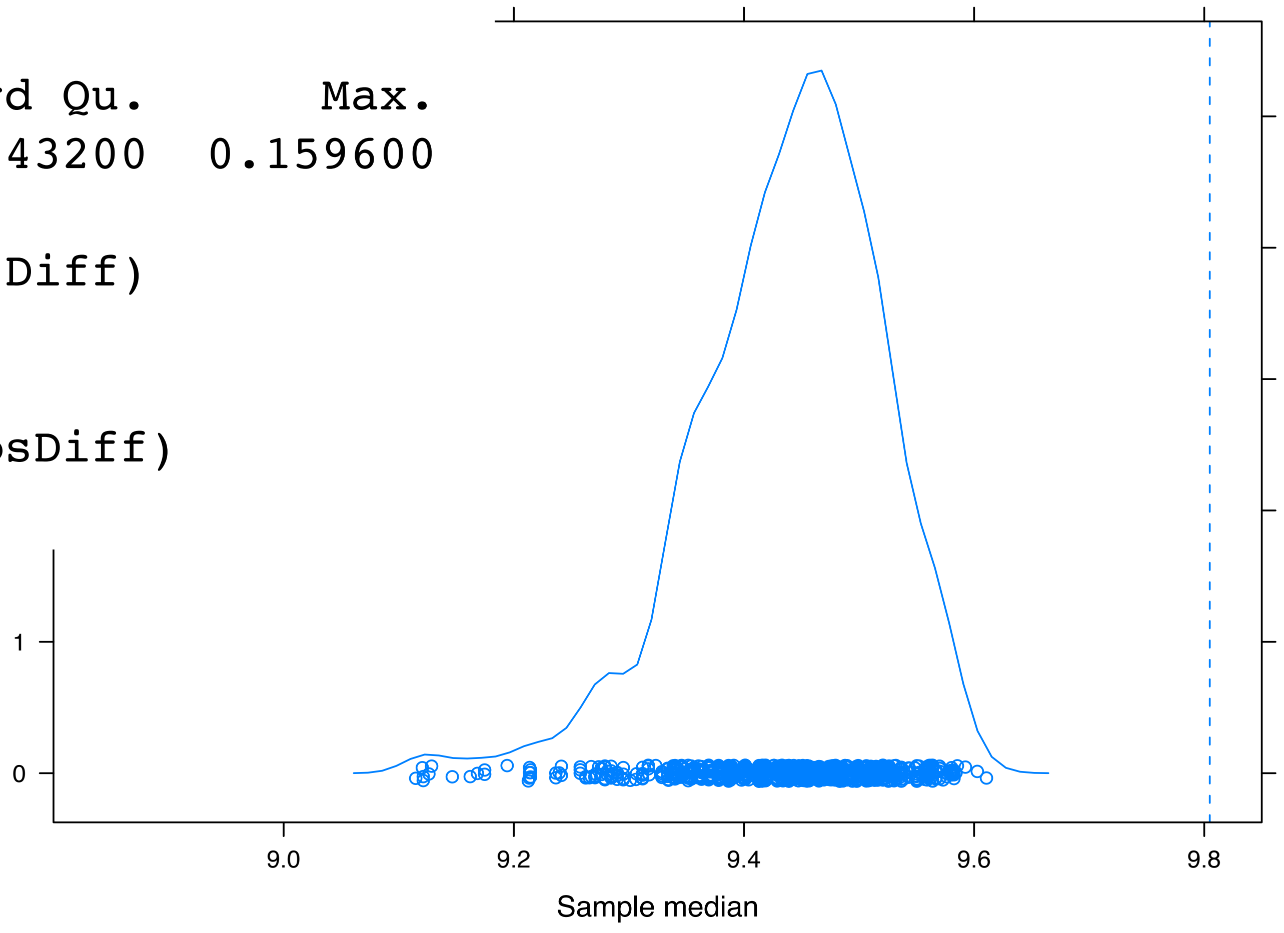
```
> ## conduct the bootstrap
> B <- 1000
> bootData2 <-
+   matrix(sample(xNull, size = B * nx, replace = TRUE),
+          nrow = nx, ncol = B)
> bootTestStat2 <- apply(bootData2, 2, median)
> densityplot(bootTestStat2,
+             panel = function(...) {
+               panel.abline(v = jMedian, lty = "dashed", col = jDefaultCol)
+               panel.densityplot(...)
+             },
+             xlab = "Sample median", xlim = c(8.8, 9.85),
+             main = "Bootstrap null distribution of sample median",
+             key = list(x = 0.05, y = 0.95, corner = c(0,1),
+               lines = list(col = jDefaultCol,
+                 lty = "dashed"),
+               text = list("observed median")))
```

```
> summary(bootTestStat2)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  9.115   9.388   9.447   9.438   9.494   9.611

> jMedian
[1] 9.804809

> binom.test(sum(x < medianNull), nx)$p.value
[1] 0.001005698

> 1/B
[1] 0.001

> (obsDiff <- abs(jMedian - medianNull))
[1] 0.3536854

> summary(bootTestStat2 - medianNull)
     Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
-0.336300 -0.063140 -0.003732 -0.013380  0.043200  0.159600

> sum(abs(bootTestStat2 - medianNull) >= obsDiff)
[1] 0

> mean(abs(bootTestStat2 - medianNull) >= obsDiff)
[1] 0
```

I conclude that

$m_{chr11} \neq m_{all}$.

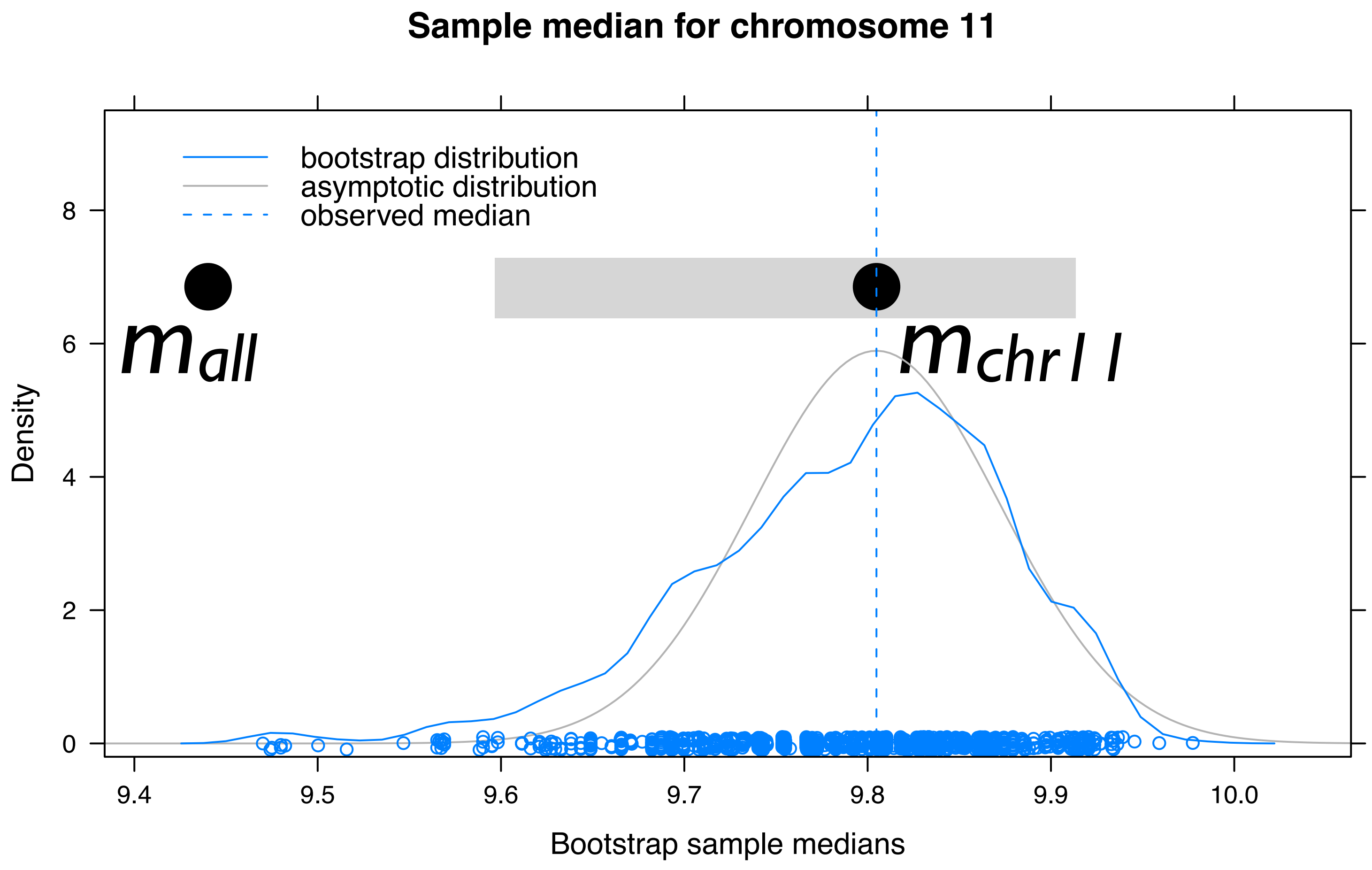
I distribution of sample median



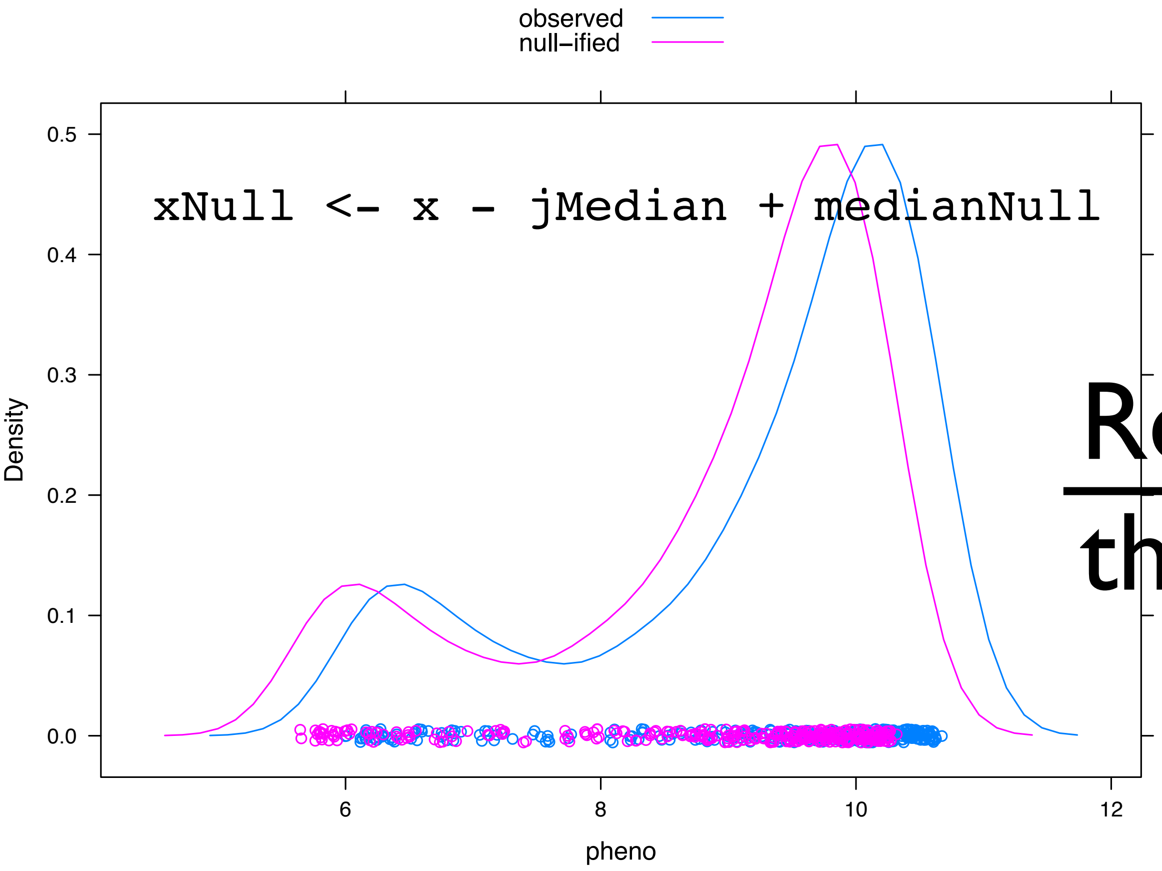
bootstrap p-value < 1/1000 = 1/B agrees well with the classical test (the binomial test)

H_0: $m_{chr11} = m_{all}$

Result: reject H_0.

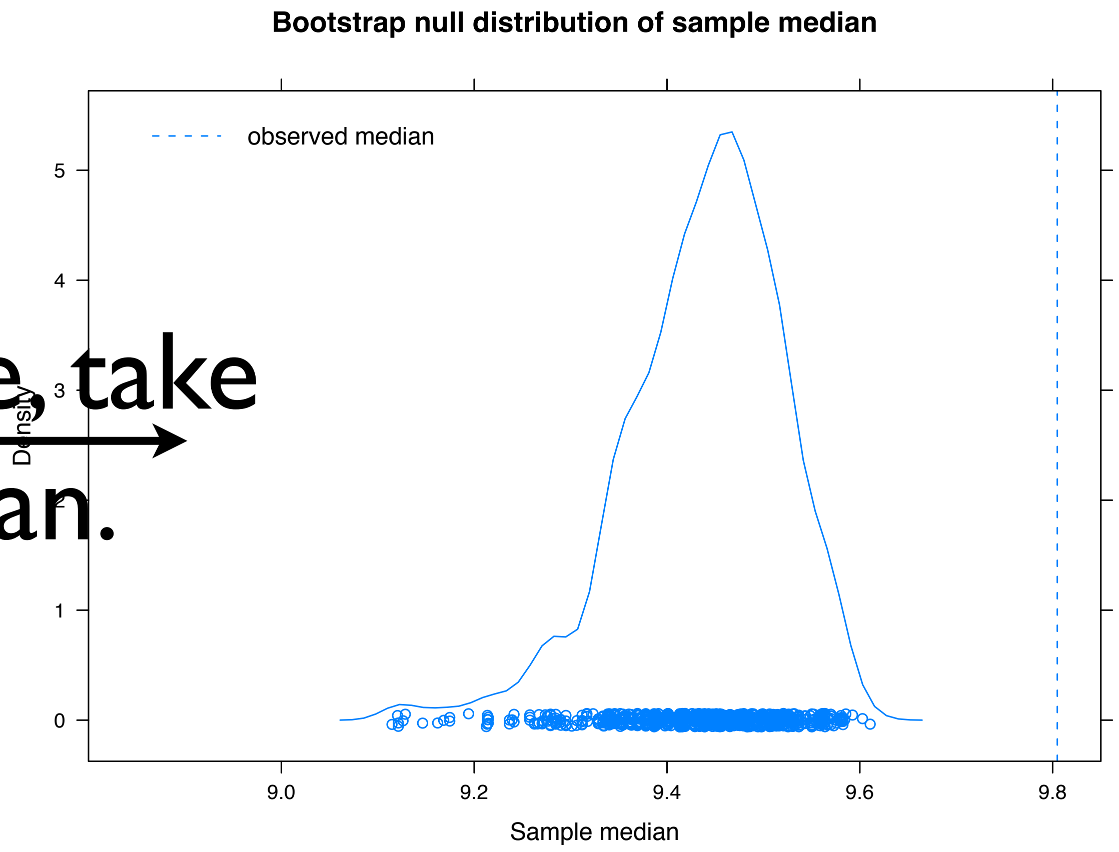


CRUCIAL: enforce H_0!

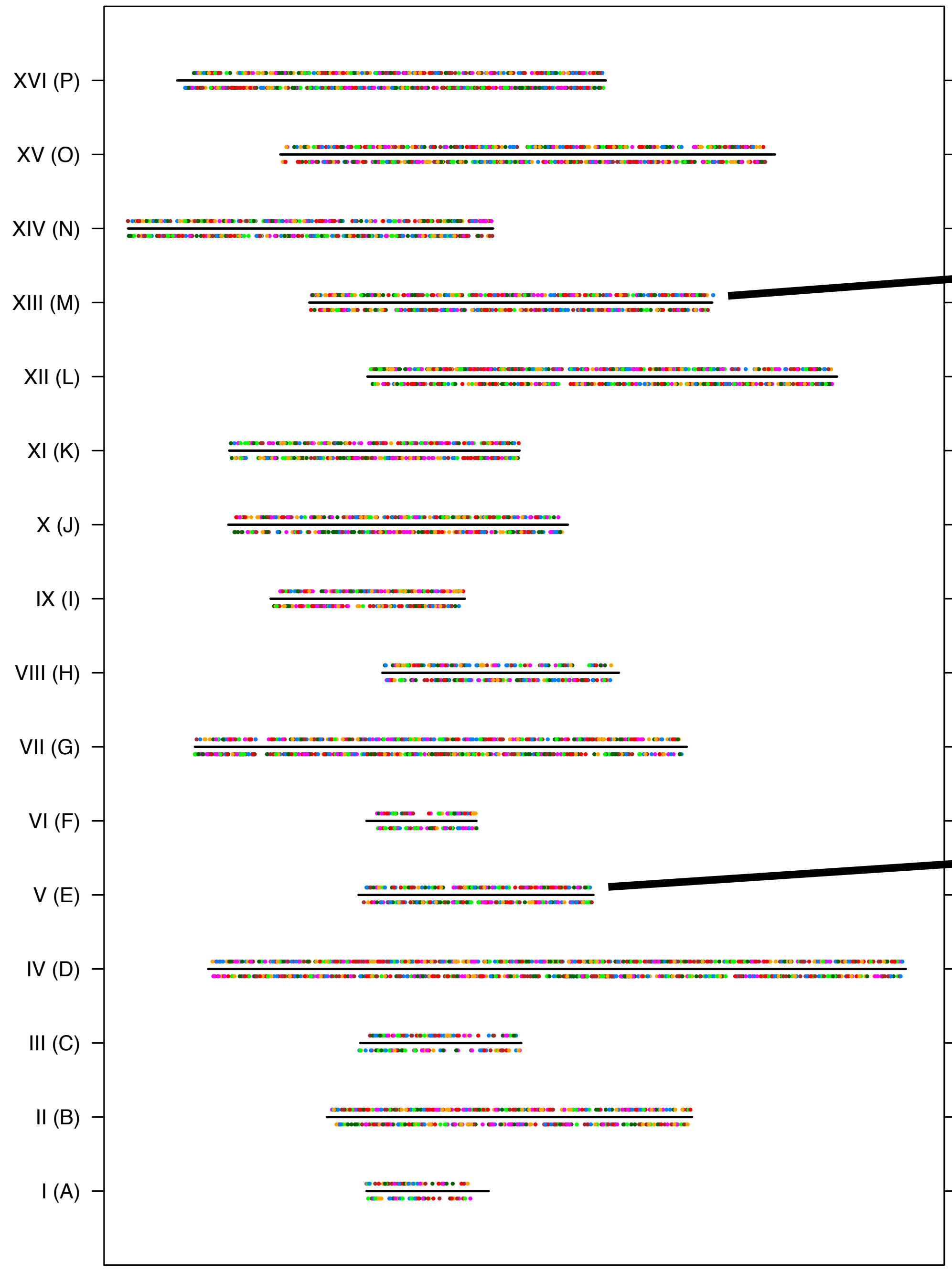


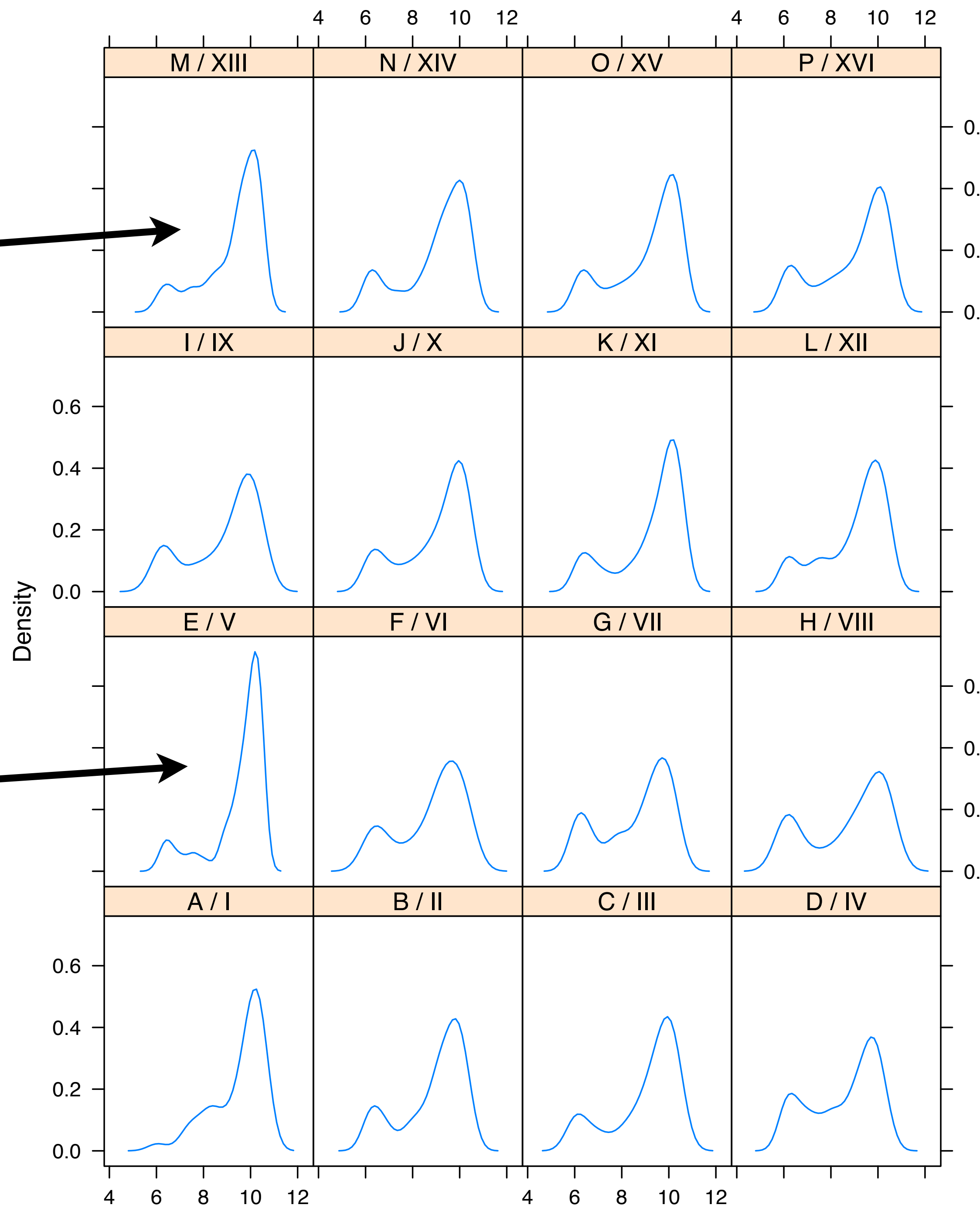
```
xNull <- x - jMedian + medianNull
```

Resample, take the median.



Do you think there are systematic differences in the phenotype distribution across the 16 chromosomes?





```
densityplot(~pheno | chromoPretty, gDat,
            plot.points = FALSE, layout = c(4,4))
```

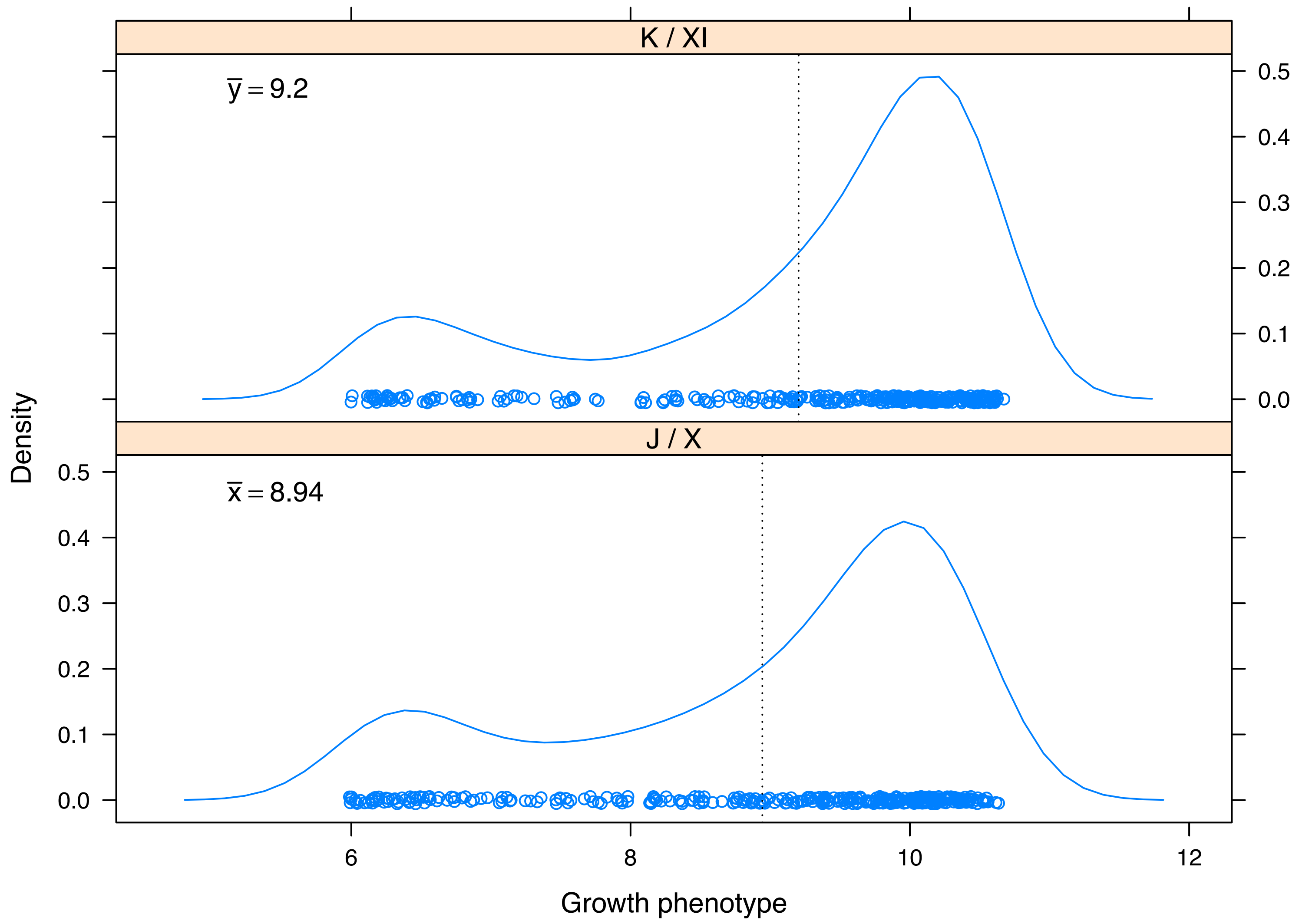
x = data observed from one chromosome, e.g. 10

y = data observed from another chromosome, e.g. 11

Regard x as a realization of X ~ F.

Regard y as a realization of Y ~ G.

F = G?



<u>Specify a null hypothesis</u> H_0: F = G (= H)

x = data observed from one chromosome
y = data observed from another chromosome

Regard x as a realization of $X \sim F$.
Regard y as a realization of $Y \sim G$.

$F = G$?

(biological questions: are the genes on different chromosomes equally important to fitness? is there a relationship between gene location and gene function or essentiality?)

Basics of a hypothesis test

- Specify a null hypothesis, H_0
- Choose a test statistic
- Determine the distribution for the test statistic under H_0
- Convert the observed test statistic into a p-value

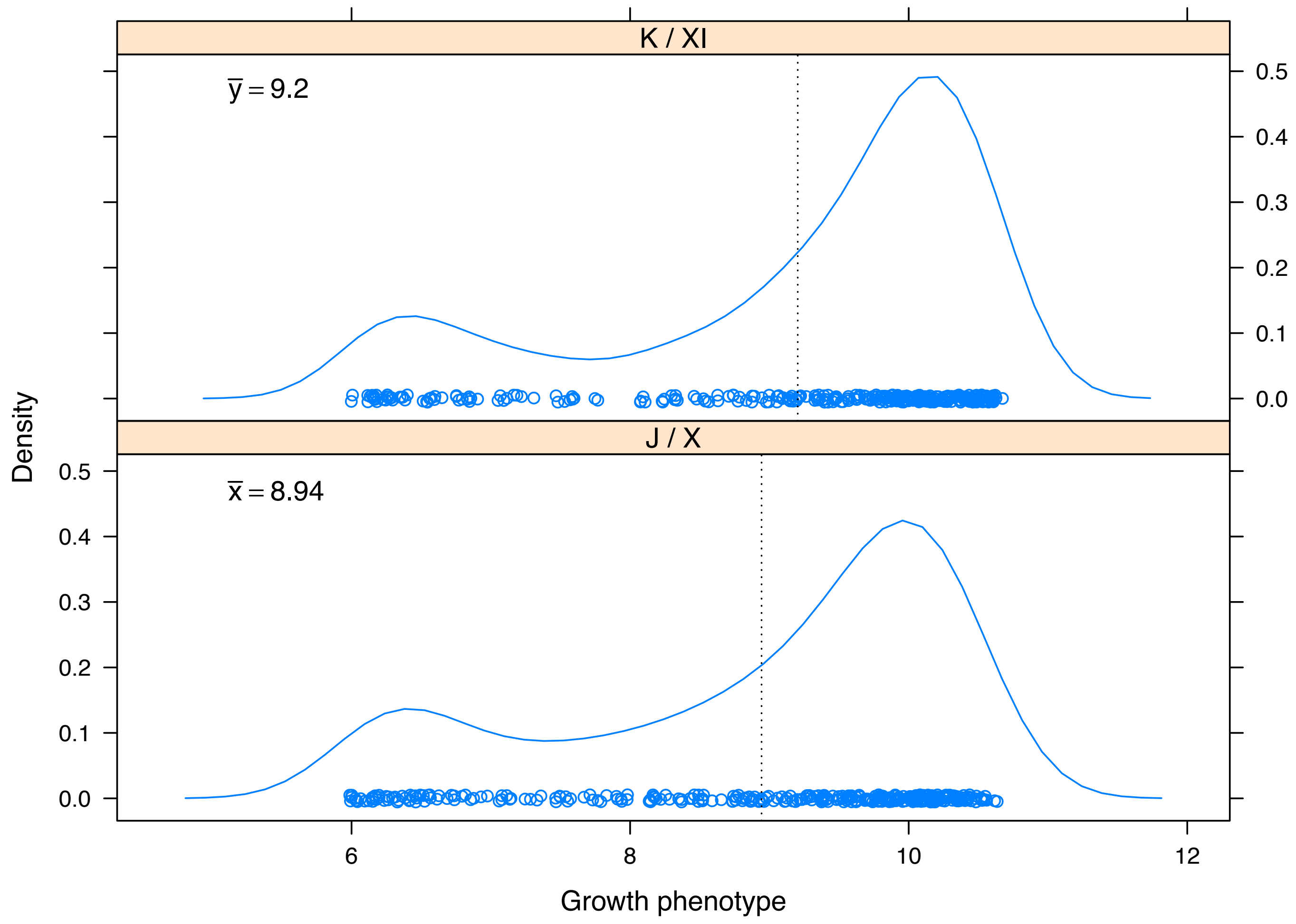
“The p-value is the probability under H_0 of observing a value of the test statistic the same or more extreme than what was actually observed.”

All of Statistics by Larry Wasserman. Springer, 2004. GoogleBooks search. via myilibrary

All of Nonparametric Statistics by Larry Wasserman. Springer, 2006. via SpringerLink | via myilibrary | GoogleBooks search.

Classical tests that address our question

- t test
- Wilcoxon test, aka Mann-Whitney here
- Kolmogorov-Smirnov test, 2 sample version
- Chi-square test of homogeneity
- I'm sure there are others ....



Critiquing the classical approach

- It can be impossible or really difficult or time-consuming to derive the null distribution of the test statistic
- In many settings, the classical hypothesis testing approach leaves us with few or no options
- If we think about hypothesis testing from first principles and we have a decent computer (and decent programming skills!), we can often empirically determine, or at least approximate very well, this null distribution and/or the p-value.
- Resampling methods, such as permutation tests and the bootstrap, take this approach.
- Key reference: *An Introduction to the Bootstrap*, by Efron and Tibshirani, Chapman & Hall / CRC, 1993. GoogleBooks search.

Null hypothesis: F = G (= H)

Possible test statistic: |avg (x) - avg (y)|

Observed value of test statistic = t

$$t = \left|\overline{x} - \overline{y}\right|$$

How much evidence does t
present against the null hypothesis?

What is the distribution of the test statistic under the null?

Under null, X and Y have same distribution. Let's call it H.

If we knew H, we could draw n_x observations from it -- call this x* -- and another n_y observations from it -- call this y*.

Compute t* = |avg x* - avg y*|.

$$t^* = \left|\bar{x}^* - \bar{y}^*\right|$$

Compute t* = |avg x* - avg y*|.

$$t^* = \left| \bar{x}^* - \bar{y}^* \right|$$

Generate B such observations t* (B large).

What proportion of the t* are as or more extreme as t? That's basically your bootstrap p-value.

Done! Sort of. We don't actually know H, though.

Here we can estimate H with an empirical distribution function.

Amalgamate x and y into one sample. Under the null, they are iid H. Give mass $1/(n_x + n_y)$ to each observation. That's the empirical distribution function. That's a decent estimate of H.

How to generate data from this estimate of H? Resample with replacement.

Choose a test statistic

Let's try this: $t = \left| \bar{x} - \bar{y} \right|$

```
> (chromoMeans <- with(kDat,
+                     tapply(pheno, chromo, mean)))
      10       11
8.943558 9.203379

> (obsTestStat <- abs(chromoMeans[1] - chromoMeans[2]))
       10
0.2598215
```

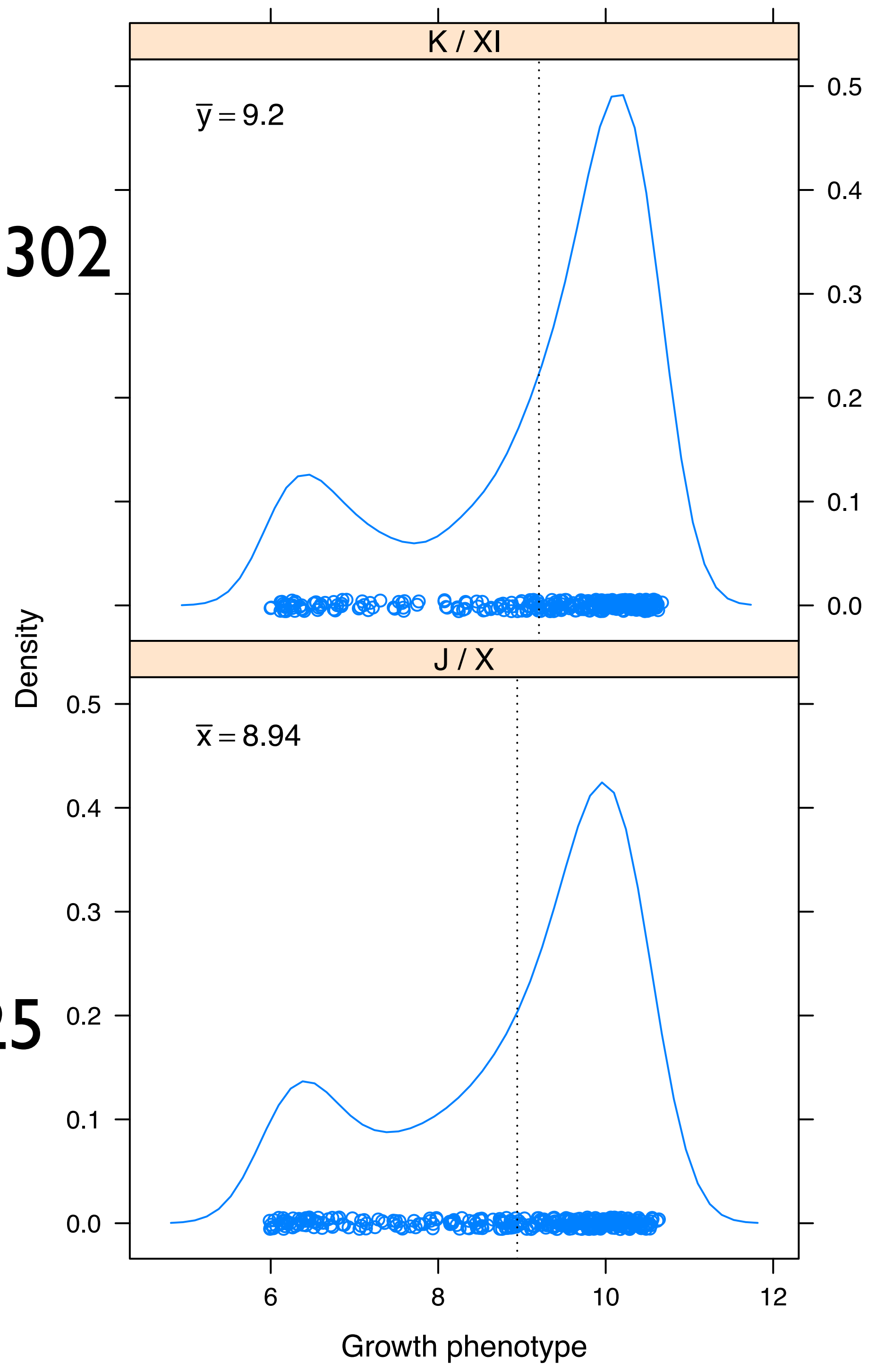
$$\bar{x} = 8.94$$

$$\bar{y} = 9.2$$

$$t = \left| \bar{x} - \bar{y} \right| = 0.26$$

$n_y = 302$

$n_x = 325$



$$t = |\bar{x} - \bar{y}| = 0.26$$

Is this "big" or "extreme" and, therefore, suggests we should reject H_0?

Ideally, we would generate lots of datasets from the (unknown) distribution H and get an empirical null distribution for this test statistic. But we don't know H ......

How to ...

Determine the distribution for the test statistic under H_0

???

Use the empirical distribution of the amalgamated data as a stand-in for the unknown H.

Generate as many “bootstrap” datasets as you like.
Compute the “bootstrap” test statistics.
Use the empirical distribution of these as your best guess at the null distribution of our test statistic.

Under null, X and Y have same distribution. Let's call it H. Estimate H with the empirical distribution of the amalgamated x's and y's.

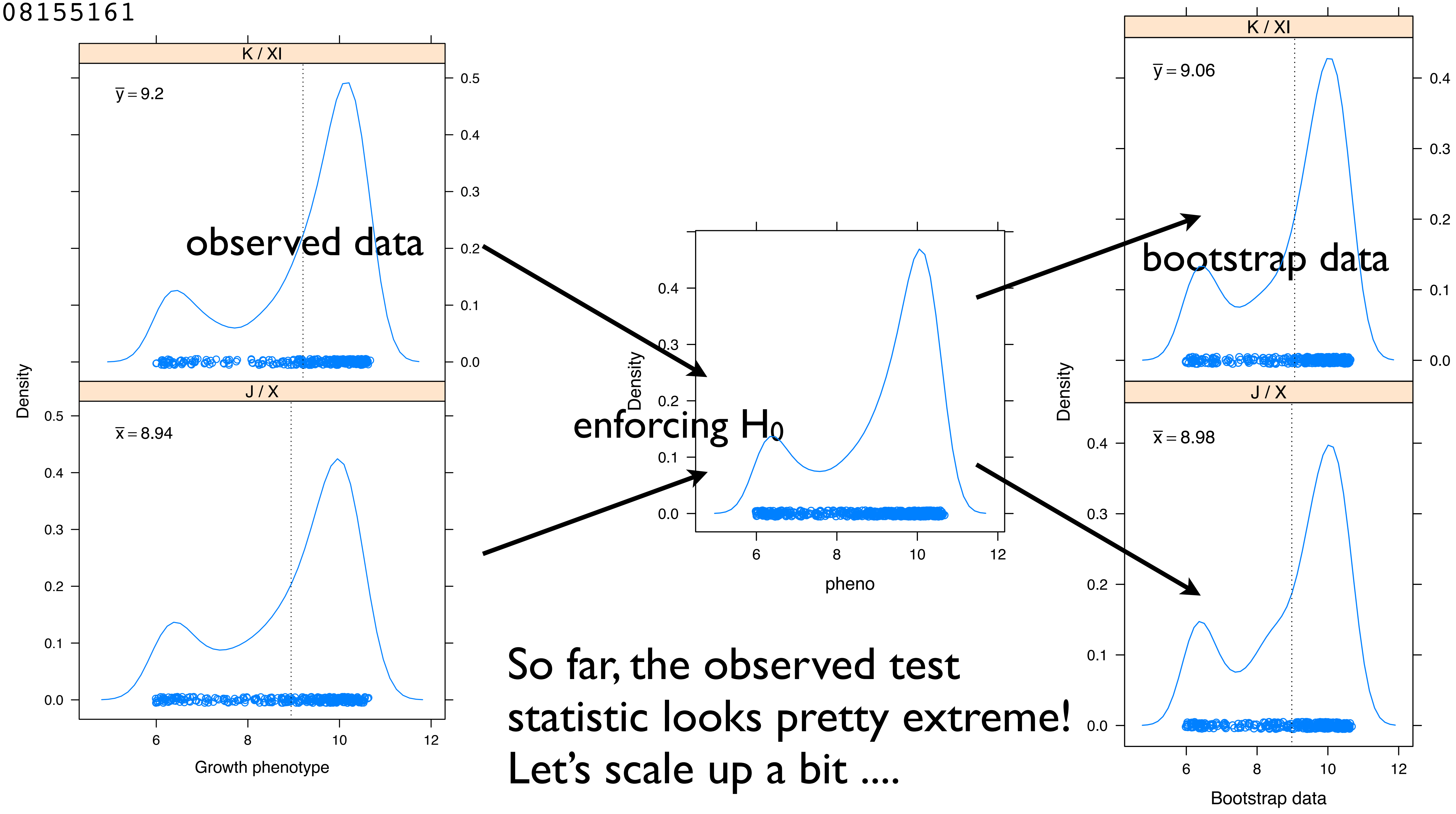
```
> ## enter the world of the null hypothesis
> ## one bootstrap sample
> set.seed(12)
> z <- kDat$pheno
> xStar <- sample(z, size = chromoCounts[1], replace = TRUE)
> yStar <- sample(z, size = chromoCounts[2], replace = TRUE)
> bootDat <- with(kDat,
+                  data.frame(pheno = c(xStar, yStar),
+                             chromo, chromoPretty))
> (bootMeans <- with(bootDat,
+                     tapply(pheno, chromo, mean)))
      10        11
8.980664 9.062216
> (bootTestStat <- abs(diff(bootMeans)))
        11
0.08155161
```

Generate bootstrap data by resampling with replacement. Draw n_x observations from it and call this x*, generate another n_y observations and call this y*.

Compute t* = |avg x* - avg y*|. $t^* = \left|\bar{x}^* - \bar{y}^*\right|$

```
> (obsTestStat <- abs(chromoMeans[1] - chromoMeans[2]))
       10
0.2598215

<snip, snip>

> (bootTestStat <- abs(diff(bootMeans)))
        11
0.08155161
```

1 bootstrap sample ("baby steps")

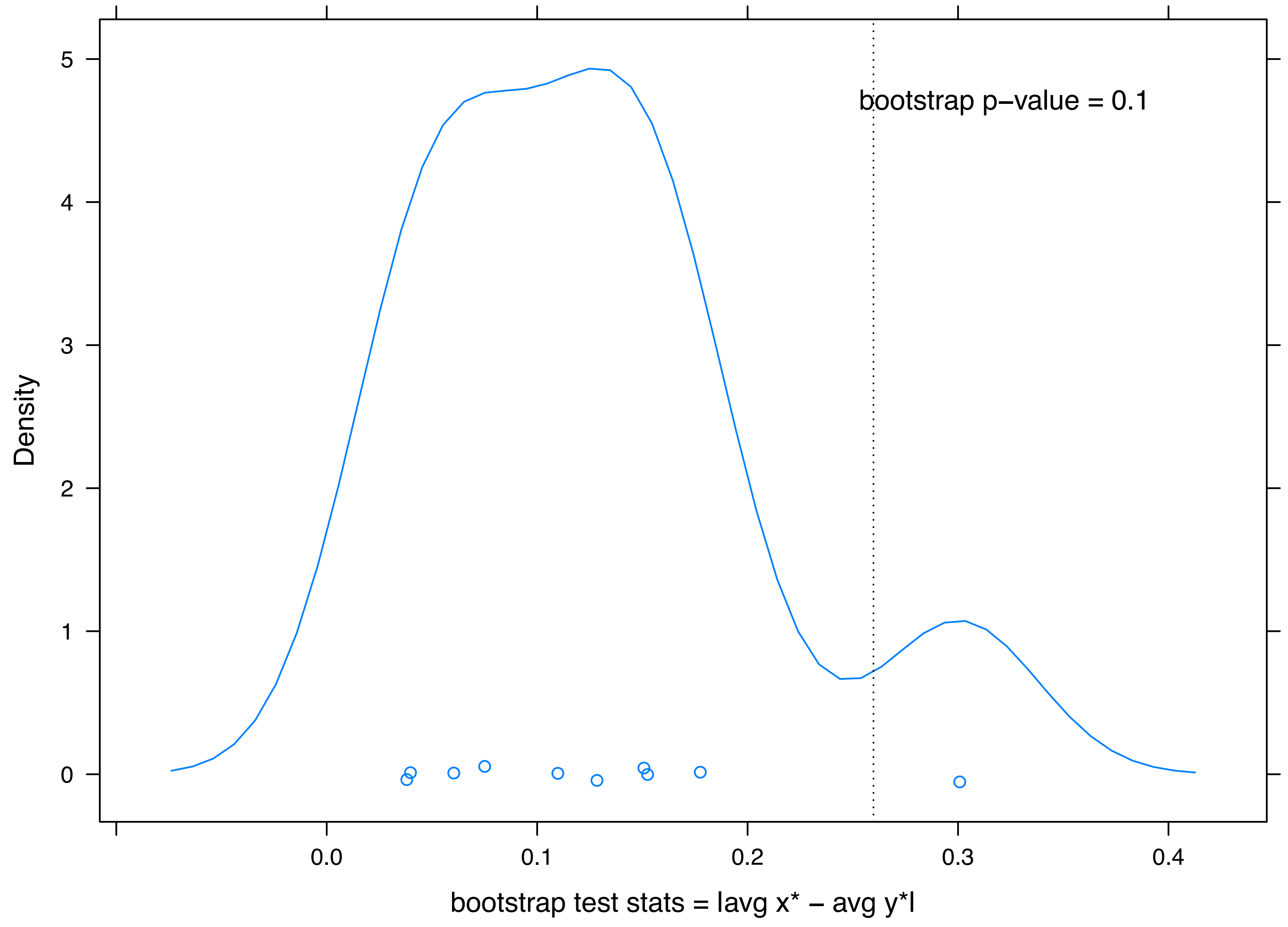


So far, the observed test statistic looks pretty extreme! Let's scale up a bit ....

```
> (obsTestStat <- abs(chromoMeans[1] - chromoMeans[2]))
       10
0.2598215

> B <- 10
> bootTestStat <- rep(NA, B)
> for(i in 1:B) {
+   xStar <- sample(z, size = chromoCounts[1], replace = TRUE)
+   yStar <- sample(z, size = chromoCounts[2], replace = TRUE)
+   bootTestStat[i] <- abs(mean(xStar) - mean(yStar))
+ }
> bootTestStat
 [1] 0.15247928 0.30081709 0.12843223 0.15073749 0.03807447 0.06039104
 [7] 0.17752854 0.07507814 0.03981151 0.10984116
> mean(bootTestStat >= obsTestStat)
[1] 0.1
```

What proportion of the t* are as or more extreme as t? That's basically your bootstrap p-value.

```
> (obsTestStat <- abs(chromoMeans[1] - chromoMeans[2]))
       10
0.2598215

...
> bootTestStat
 [1] 0.23677776 0.21074474 0.16380568 0.13258165 0.01663695 0.07176389
 [7] 0.09824504 0.20745668 0.11928580 0.27759690
> mean(bootTestStat >= obsTestStat)
[1] 0.1
```



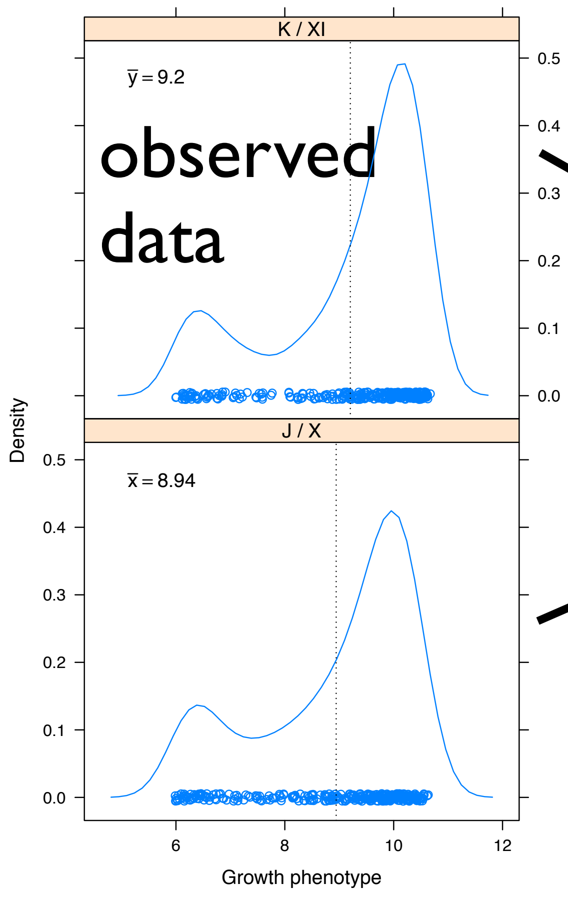
What proportion of the t* are as or more extreme as t? That's basically your bootstrap p-value.

Are you drunk with power now?

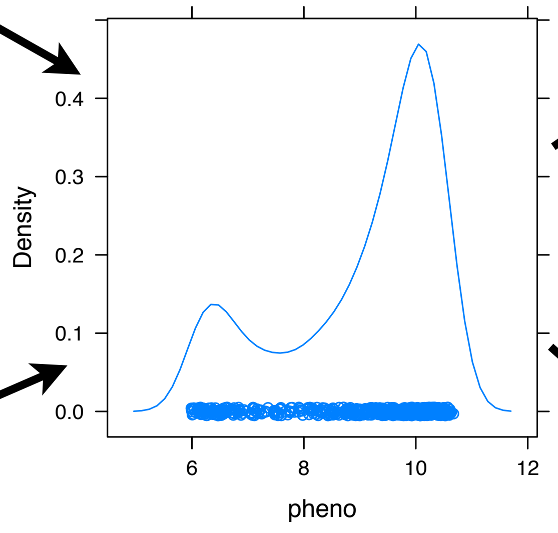
- You really can pursue hypothesis testing now with whatever test statistic you find to be most relevant.
- Or ... you can work confidently with a "classical" test statistic without making assumptions about normality or 'n large' ...
- Let's try a much larger value of B.

bootstrap data

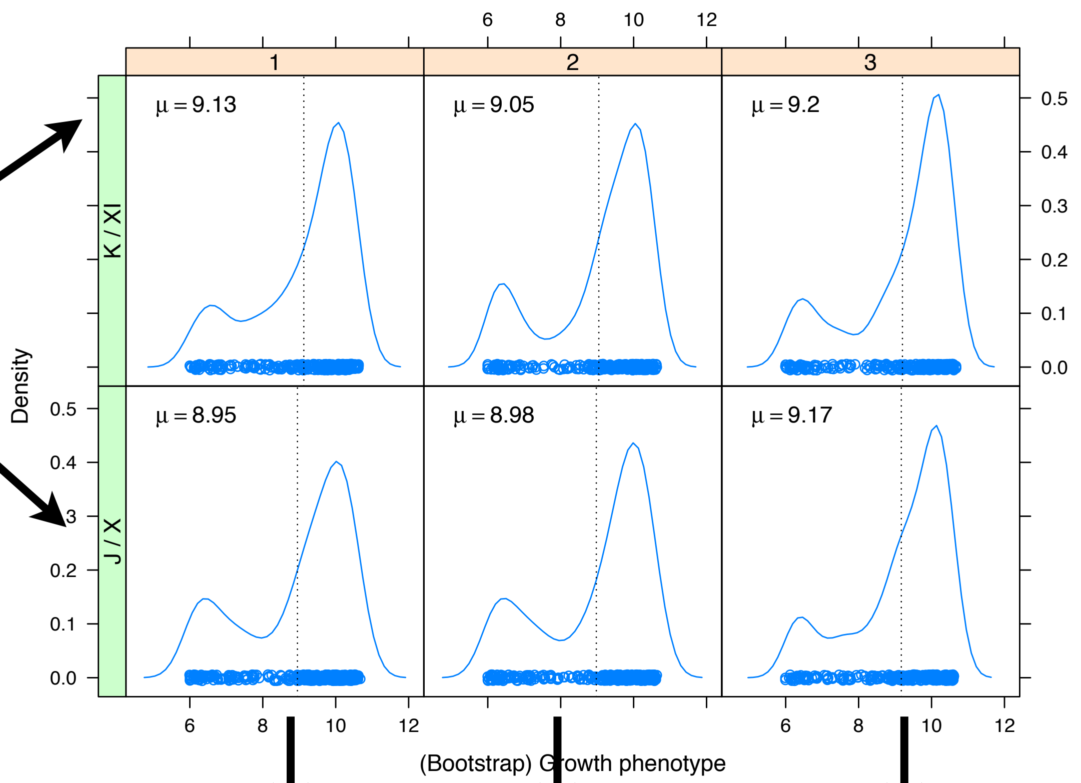
observed data



$$t = |\bar{x} - \bar{y}| = 0.26$$



enforce H_0



and 9,997 more like this ....

$$t^* = |\bar{x}^* - \bar{y}^*| = 0.17$$

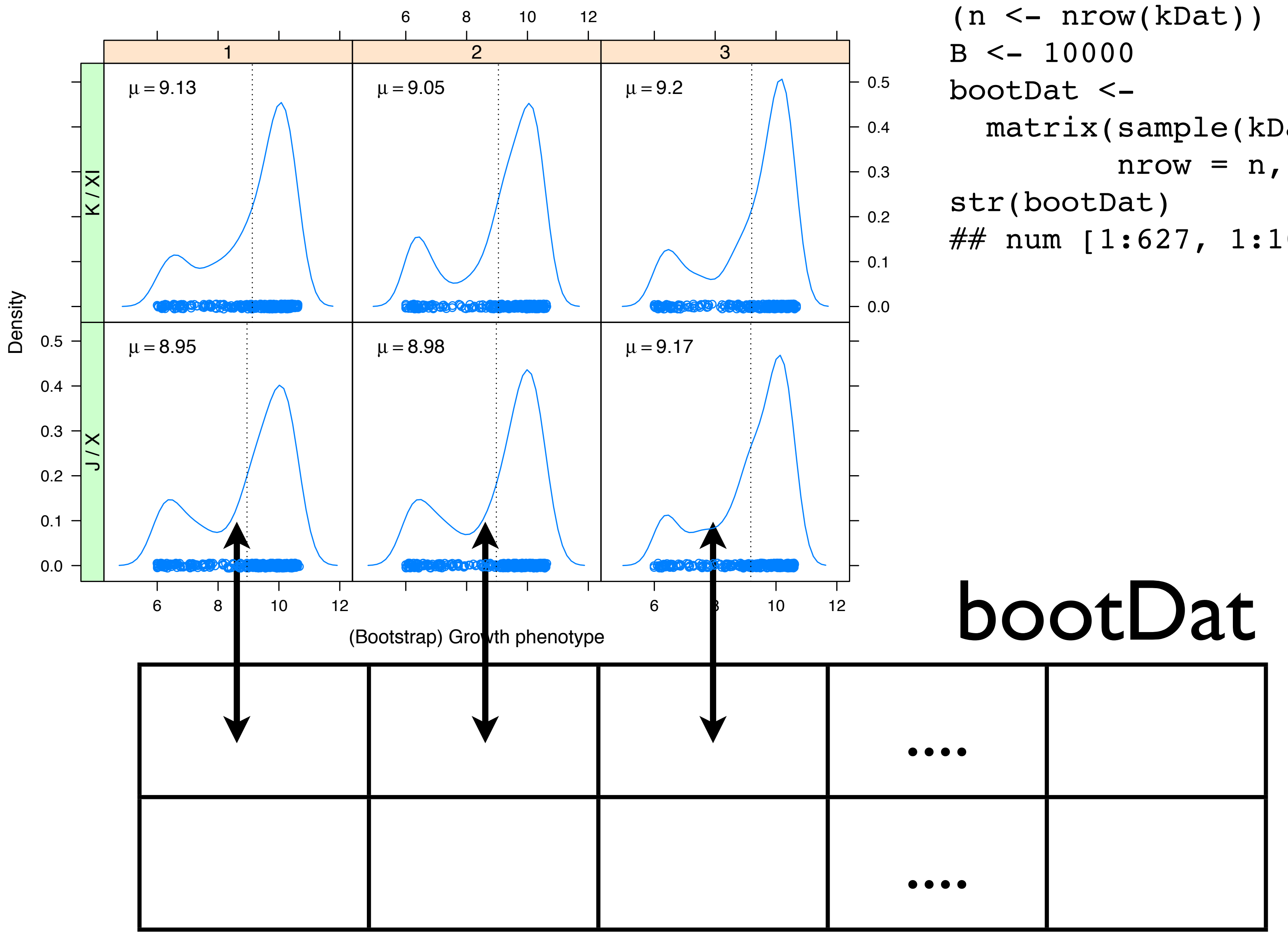
$$t^* = |\bar{x}^* - \bar{y}^*| = 0.07$$

$$t^* = |\bar{x}^* - \bar{y}^*| = 0.03$$

```
(n <- nrow(kDat))                          # 627 obs
B <- 10000
bootDat <-
  matrix(sample(kDat$pheno, size = B * n, replace = TRUE),
         nrow = n, ncol = B)
str(bootDat)
## num [1:627, 1:10000] 10.08 10.07 10.03 9.26 8.28 ...
bootTestStats <-
  apply(bootDat, 2, computeAbsDifferenceOfMeans, jFact = kDat$chromo)
```

No explicit loops!

B = 10,000 bootstrap samples



```
(n <- nrow(kDat))                          # 627 obs
B <- 10000
bootDat <-
  matrix(sample(kDat$pheno, size = B * n, replace = TRUE),
         nrow = n, ncol = B)
str(bootDat)
## num [1:627, 1:10000] 9.73 6.15 9.96 9.4 9.46 ...
```

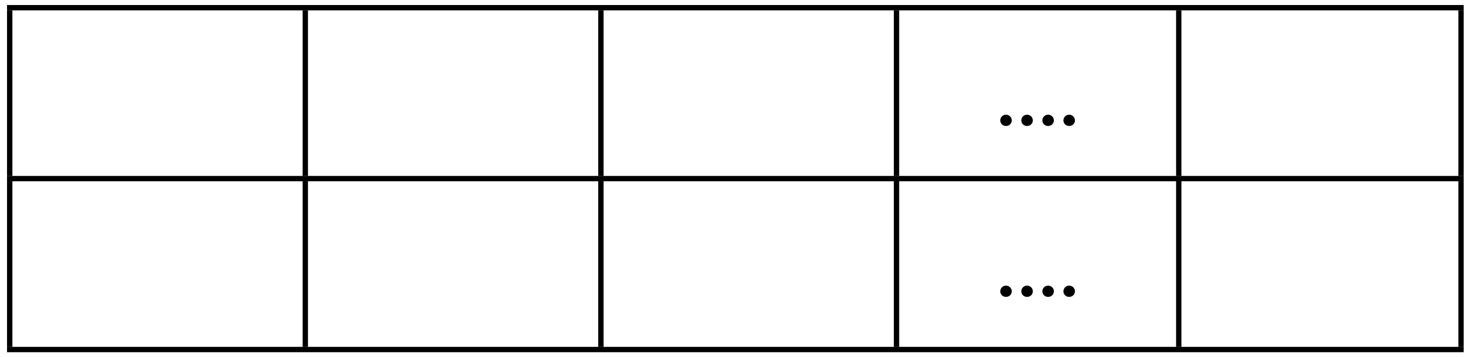
bootDat

each column is 1 bootstrap sample

n = 325 + 302 = 627 rows by B = 10,000 columns

You can often generate all the bootstrap data at once, i.e. no need to explicitly loop for b = 1 to B.

bootDat



each column is 1 bootstrap sample

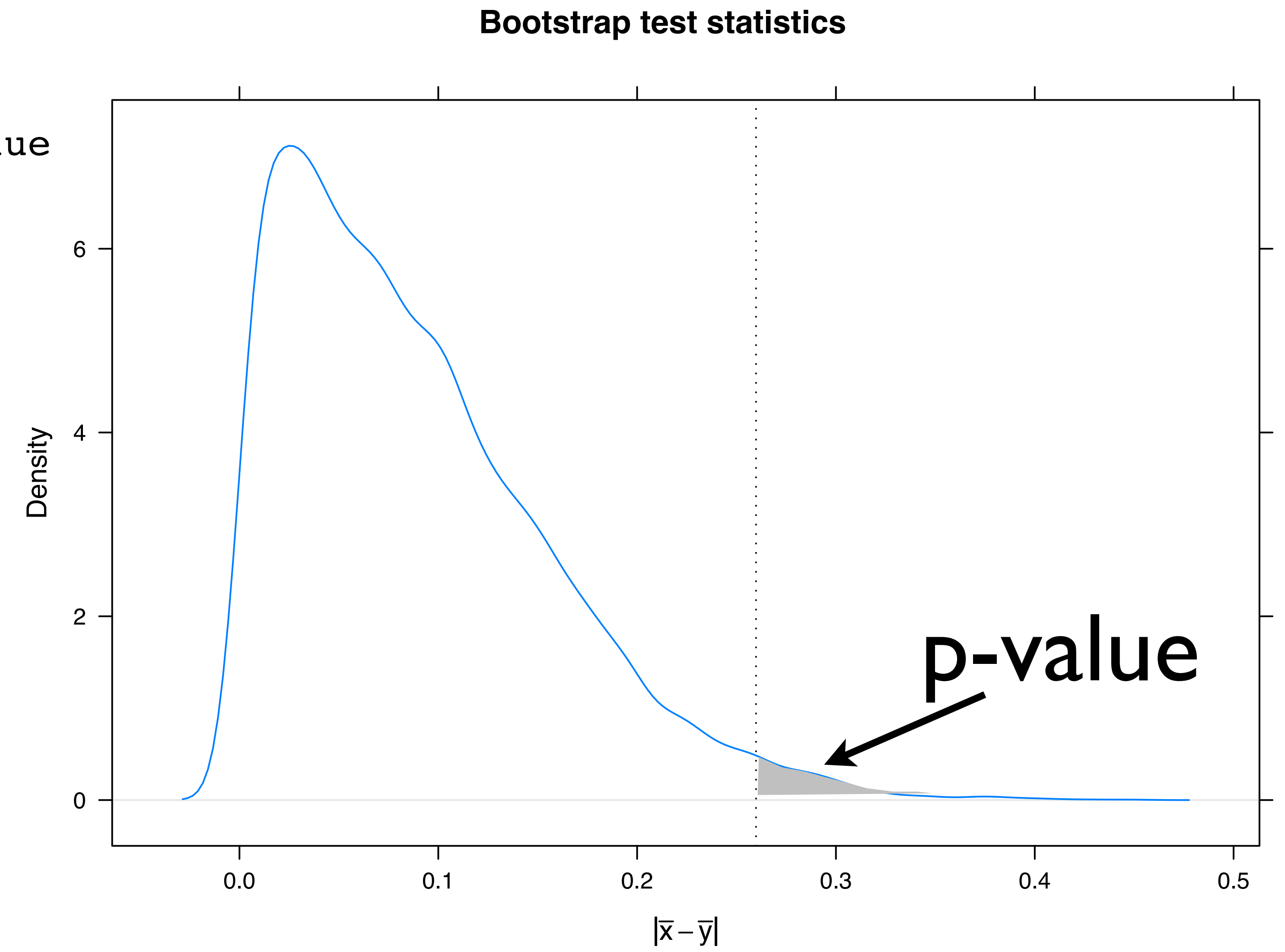
n = 325 + 302 = 627 rows by B = 10,000 columns

```
## function to compute my test statistic
computeAbsDifferenceOfMeans <- function(jResp, jFact) {
  groupMeans <- tapply(jResp, jFact, mean)
  return(abs(groupMeans[1] - groupMeans[2]))
}

bootTestStats <-
  apply(bootDat, 2, computeAbsDifferenceOfMeans, jFact = kDat$chromo)
```

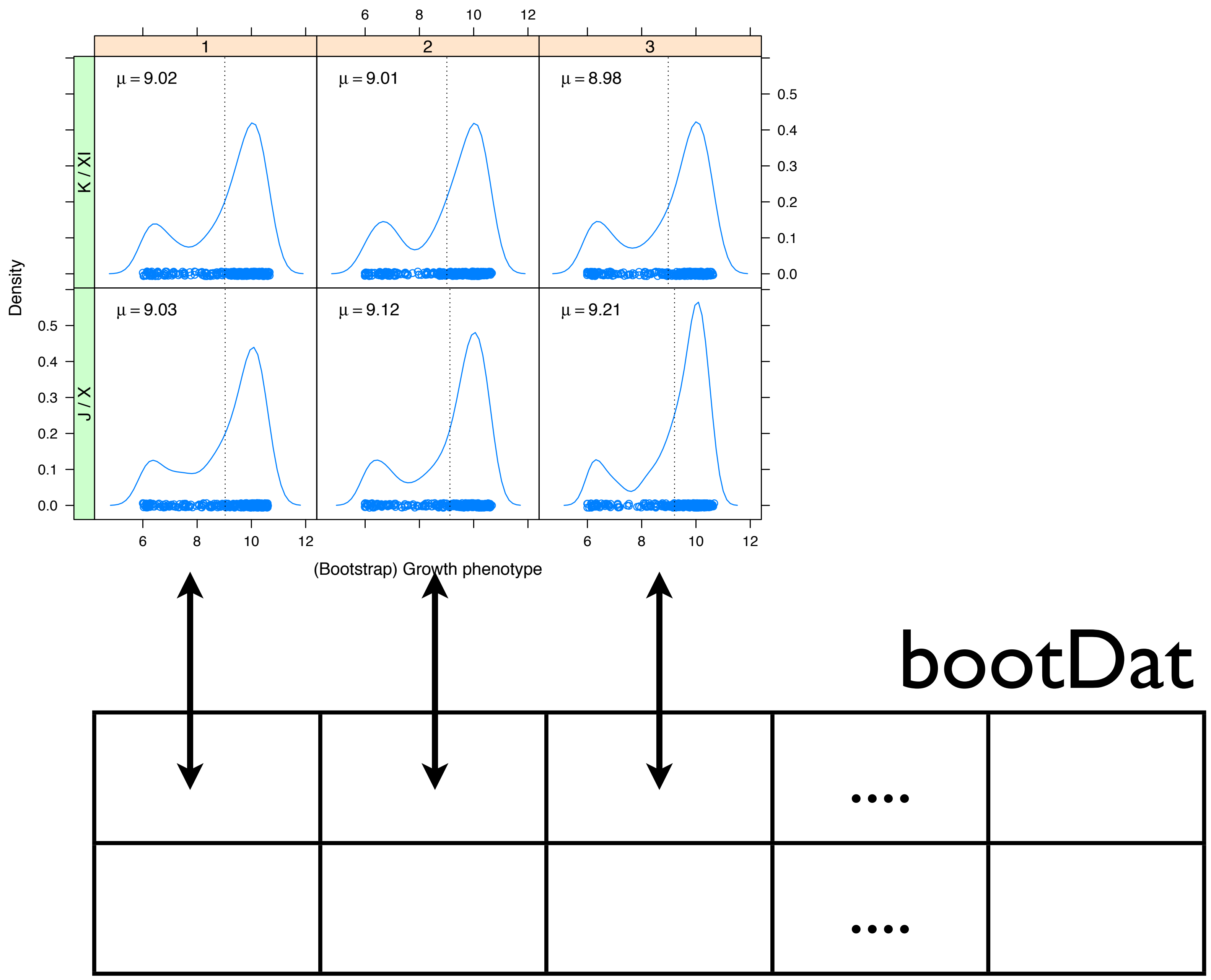
“To each column of bootData, apply my function to compute the test statistic.”

You can then use an apply function to compute the test statistic for each bootstrap data set. No need to explicitly loop for b = 1 to B.

```
> bootTestStats <-
+         apply(bootDat, 2, computeAbsDifferenceOfMeans, jFact = kDat$chromo)

> densityplot( ~ bootTestStats,
+             xlab = expression(group("|", bar(x) - bar(y),"|")),
+             main = "Bootstrap test statistics",
+             plot.points = FALSE, n = 200, ref = TRUE,
+             panel = function(x, ...) {
+               panel.densityplot(x, ...)
+               panel.abline(v = obsTestStat, lty = 'dotted')
+             })

> ## bootstrap p-value
> mean(bootTestStats >= obsTestStat)
[1] 0.0172

> t.test(pheno ~ chromo, kDat)$p.value
[1] 0.01940612

> wilcox.test(pheno ~ chromo, kDat)$p.value
[1] 0.001156313
```



Bootstrap p-value is awfully close to that from Welch's t-test. That's comforting. Wilcoxon p-value is yet smaller -- probably due to greater efficiency in large samples w/ non-normal data.

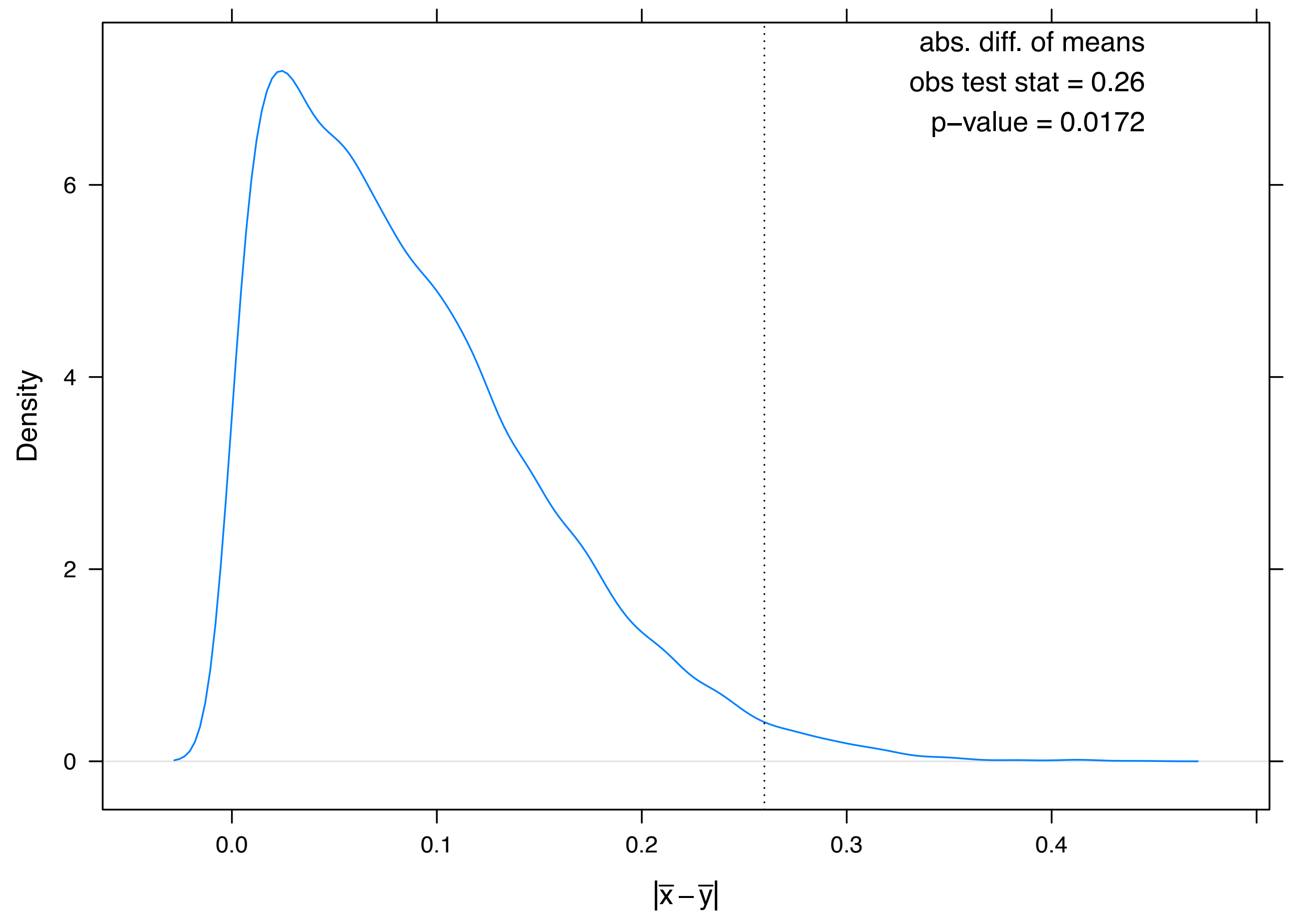


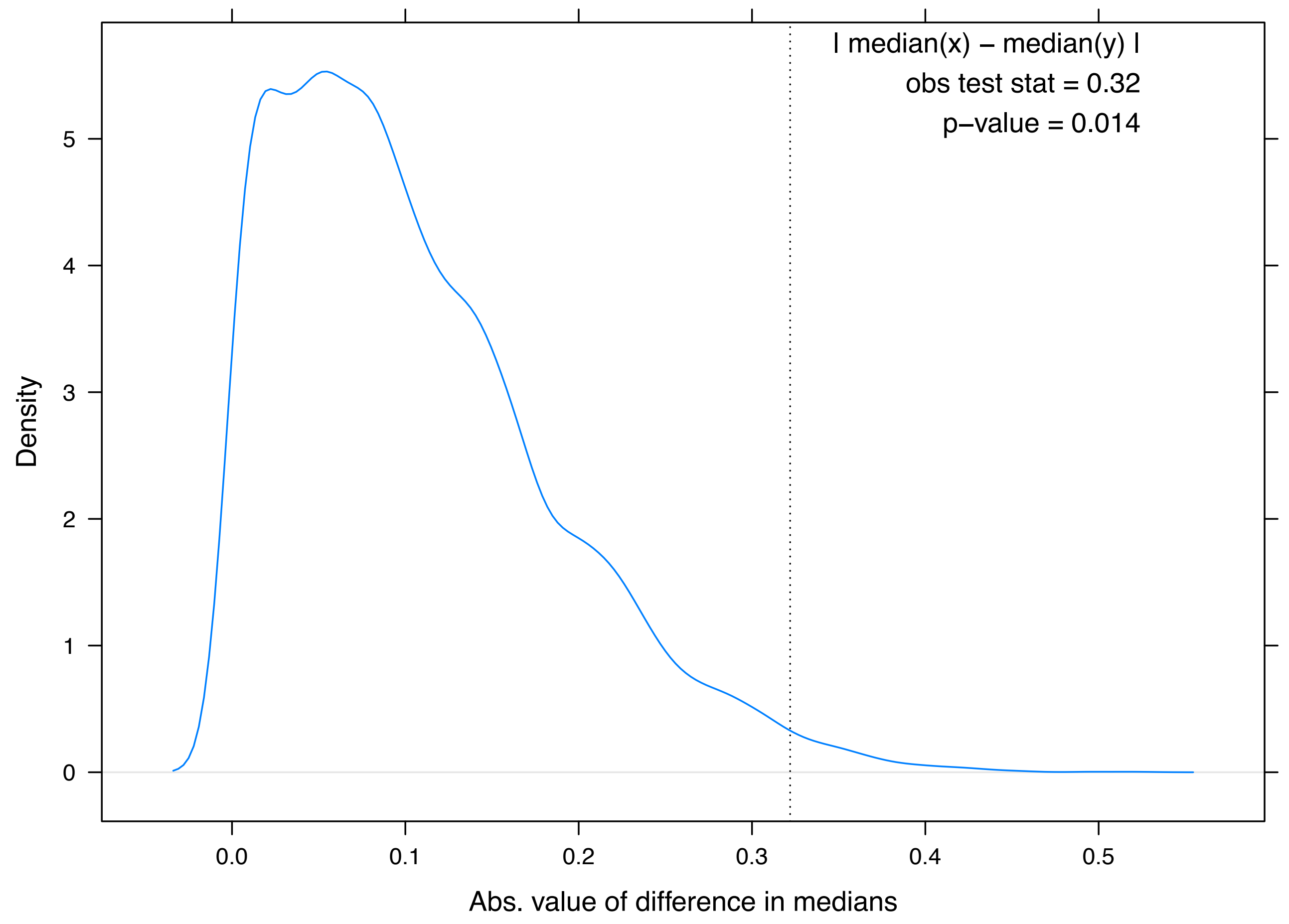
bootDat

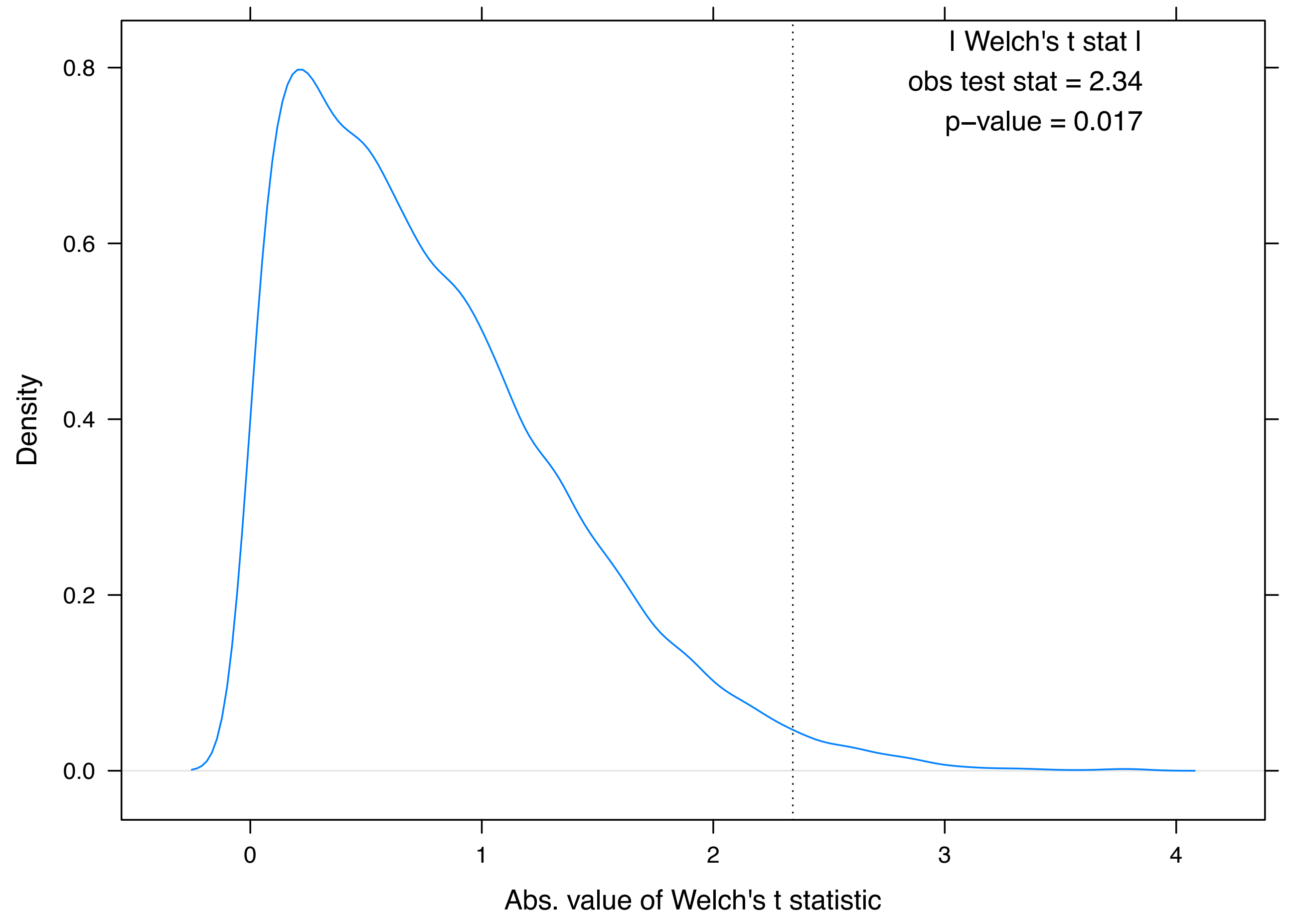
each column is 1 bootstrap sample

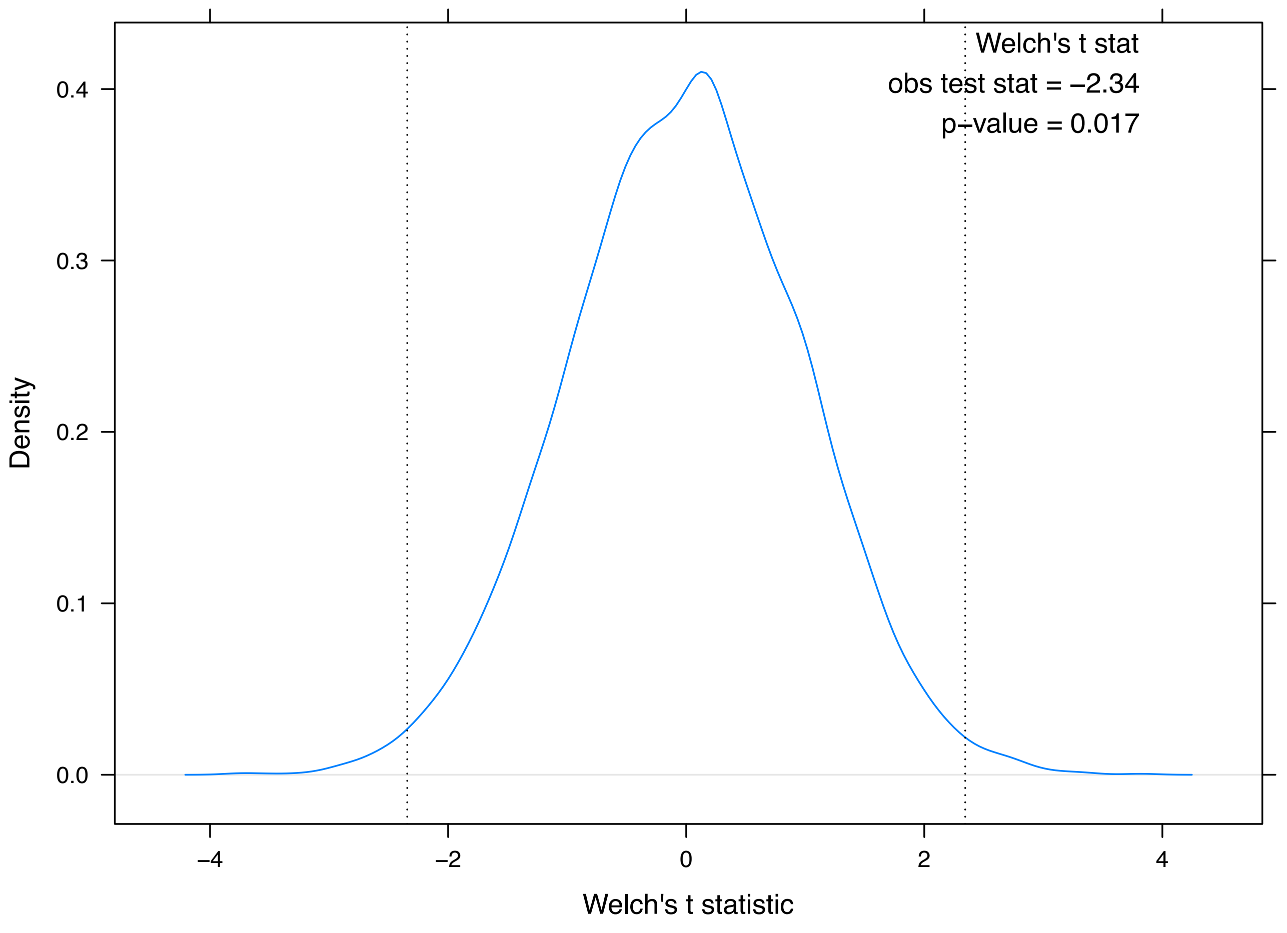
n = 325 + 302 = 627 rows by B = 10,000 columns

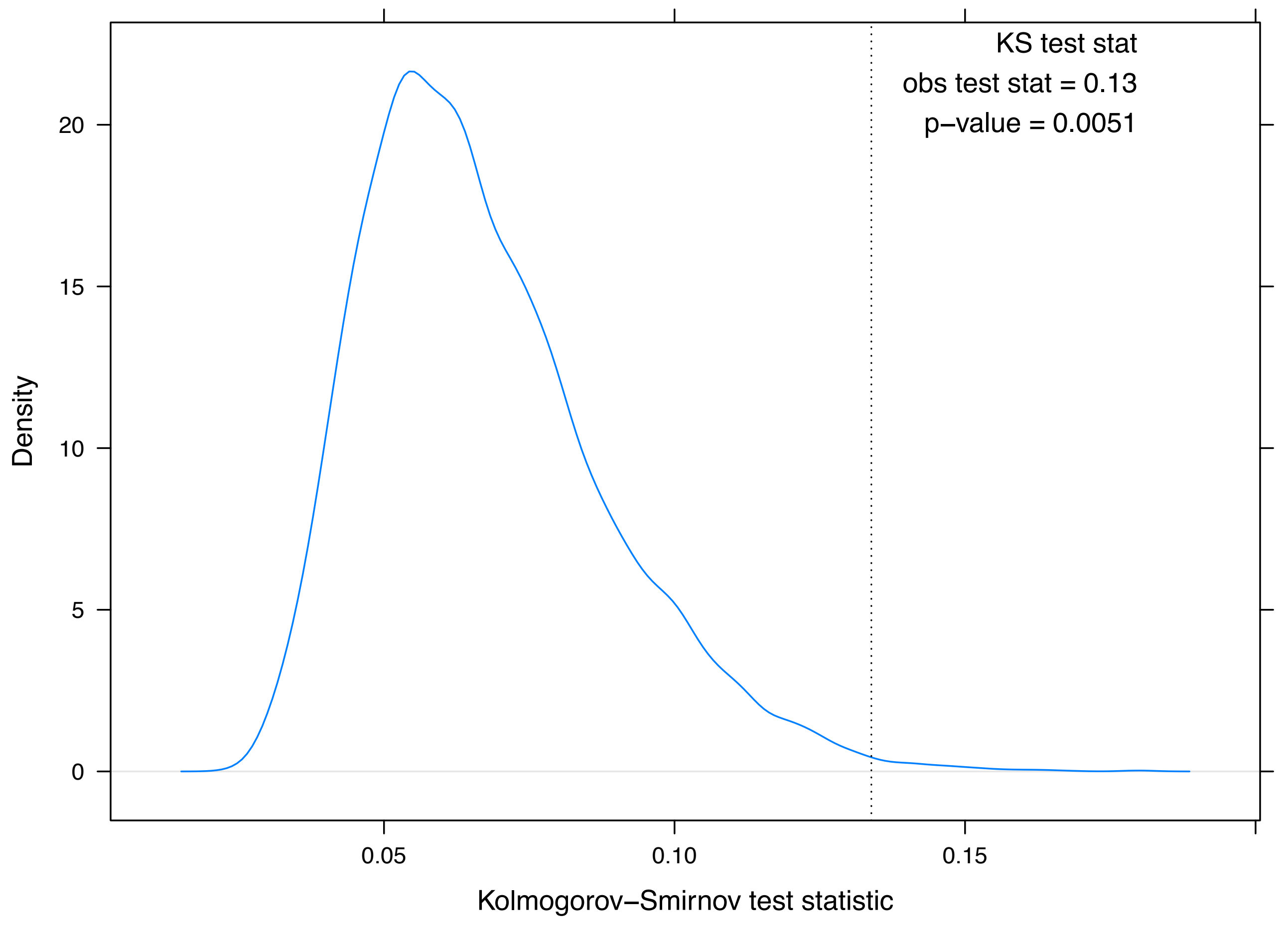
We can use these bootstrap datasets to get empirical null distributions for the other test statistics we've been considering. Neat chance to check results using classical theory against those from brute force computing.

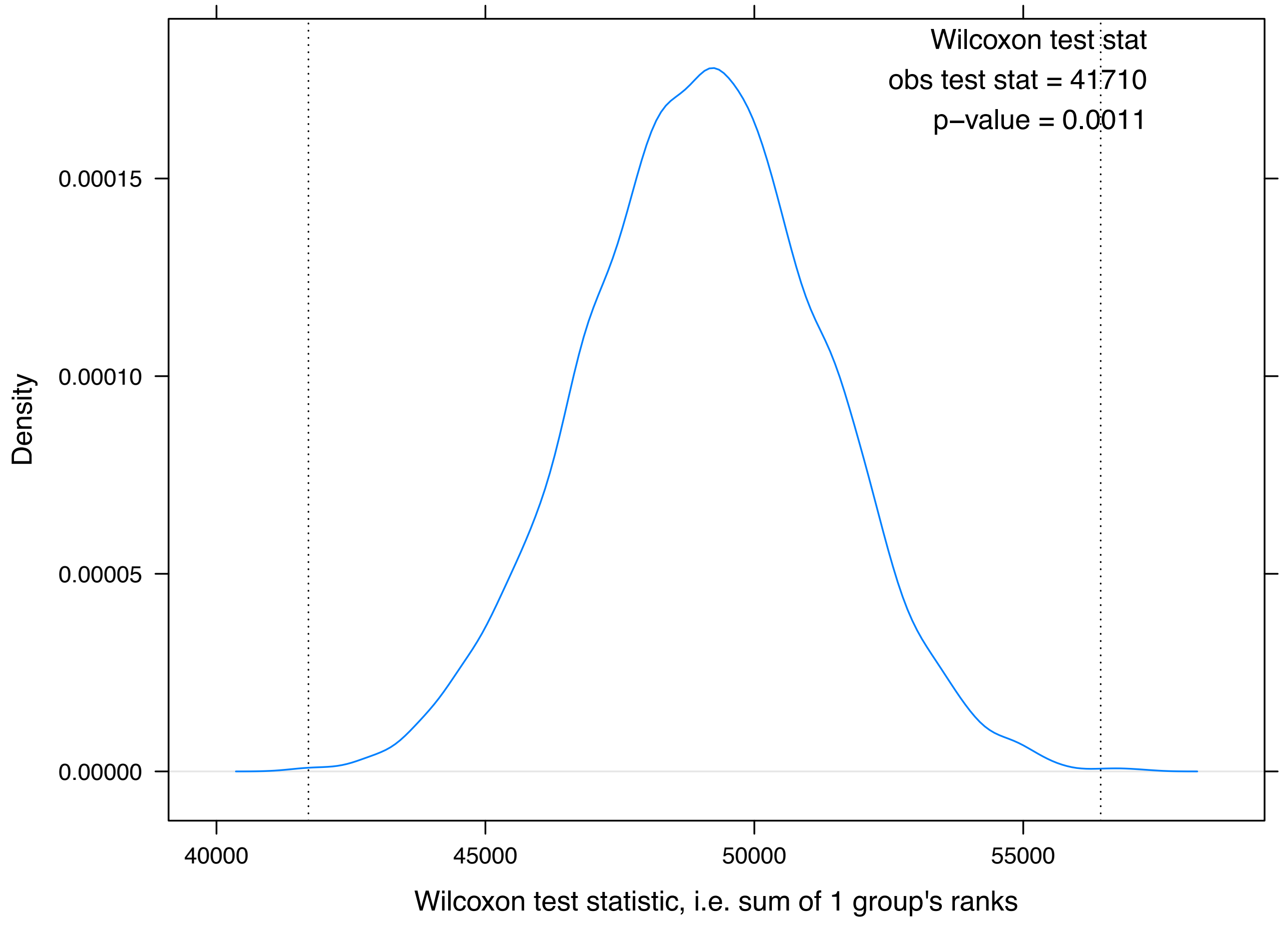

abs. diff. of means
obs test stat = 0.26
p−value = 0.0172
Density
0
2
4
6
0.0
0.1
0.2
0.3
0.4
$|\bar{x} - \bar{y}|$


| median(x) − median(y) |
obs test stat = 0.32
p−value = 0.014
Density
0
1
2
3
4
5
0.0
0.1
0.2
0.3
0.4
0.5
Abs. value of difference in medians


| Welch's t stat |
obs test stat = 2.34
p−value = 0.017
Density
0.0
0.2
0.4
0.6
0.8
0
1
2
3
4
Abs. value of Welch's t statistic


Welch's t stat
obs test stat = −2.34
p−value = 0.017
Density
0.0
0.1
0.2
0.3
0.4
−4
−2
0
2
4
Welch's t statistic


KS test stat
obs test stat = 0.13
p−value = 0.0051
Density
0
5
10
15
20
0.05
0.10
0.15
Kolmogorov−Smirnov test statistic


Wilcoxon test stat
obs test stat = 41710
p−value = 0.0011
Density
0.00000
0.00005
0.00010
0.00015
40000
45000
50000
55000
Wilcoxon test statistic, i.e. sum of 1 group's ranks

test statistic	observed value	“classical” p-value	bootstrap p-value
$t = \lvert \bar{x} - \bar{y} \rvert$	0.26	NA	0.0172
$t = \lvert median(x) - median(y) \rvert$	0.322	NA	0.0140
|Welch’s t statistic |	2.344	NA	0.0170
Welch’s t statistic	-2.344	0.0194	0.0170
Kolmogorov-Smirnov	0.134	0.0073	0.0051
Wilcoxon	41710	0.0012	0.0011

Why the bootstrap is important

“If we choose a statistic more complicated than <sthgSimple> or a distribution less tractable than <sthgFriendly>, then no amount of mathematical cleverness will yield a simple formula.

Because of such limitations, pre-computer statistical theory focused on a small set of distributions and a limited class of statistics.

Computer-based methods like the bootstrap free the statistician from these constraints.”

p. 60-1 of Efron & Tibshirani

With power comes responsibility

"This is not all pure gain.

Theoretical formulas ... can help us understand a situation in a different way than the numerical output of a bootstrap program.

It pays to remember that methods like the bootstrap free the statistician to look *more closely* at the data, without fear of mathematical difficulties, not *less closely*."

p. 60-1 of Efron & Tibshirani