

STAT 545A

Class meeting #11

Monday, October 15, 2012

Dr. Jennifer (Jenny) Bryan

Department of Statistics and Michael Smith Laboratories

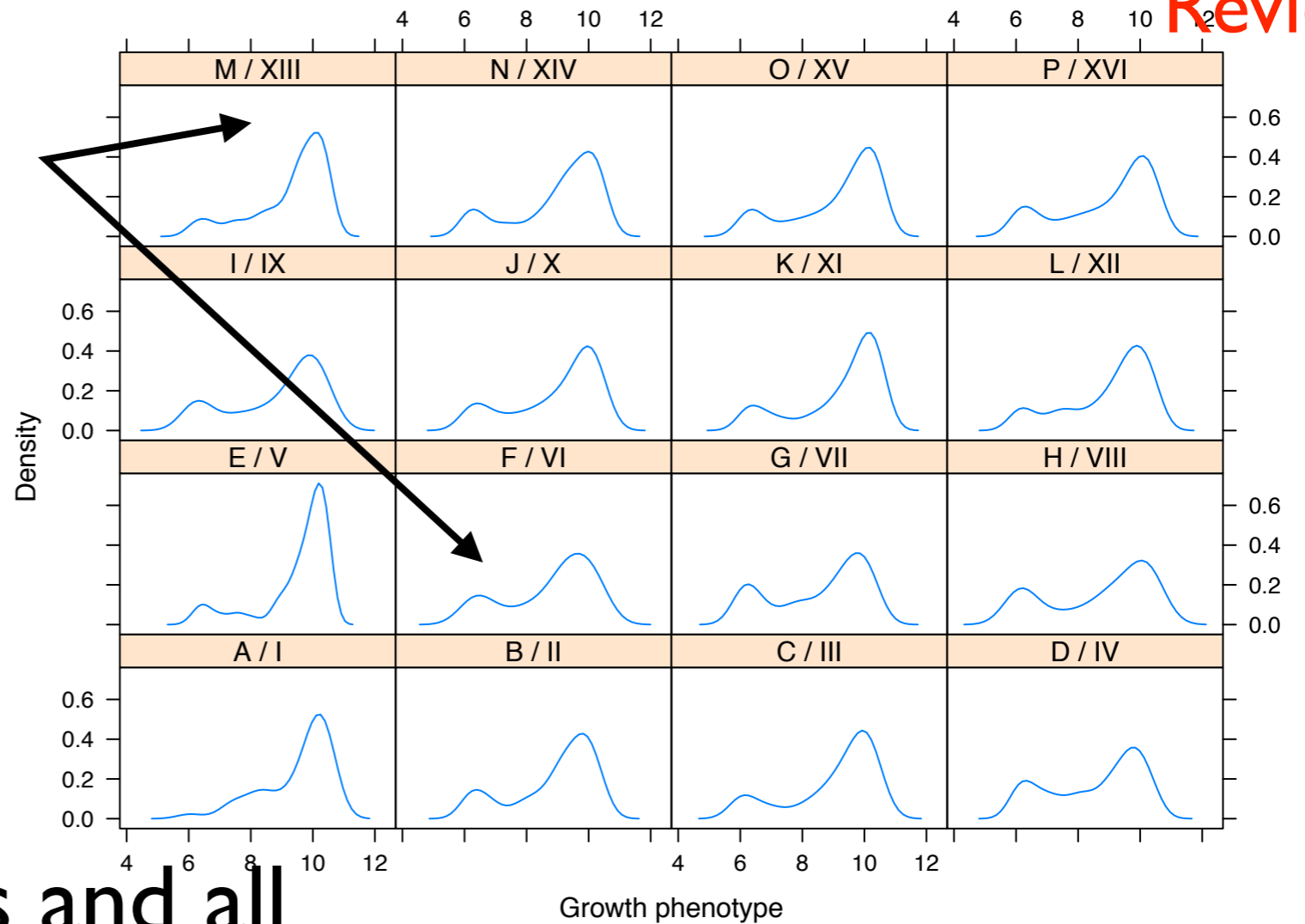
Review of last class

Worked with yeast growth data. For each gene in the genome, we observed yeast growth when that gene was deleted. Quantitative response variable is a readout for growth. Categorical explanatory variable was chromosome.

We conducted lots of “two groups” tests, assessing whether the distribution of growth is same or different for gene (deletions) on different chromosomes.

Presented 120 p-values -- from all possible pairs of chromosomes -- as a heatmap. Upgrades included color scheme, supplementing w/ original density plots, rational ordering of the chromosome factor, and probit transformation of the p-values.

Same underlying data-generating distribution?
Assess with a “two groups” test.



Consider various tests and all possible pairwise comparisons (120, in this case).

```
> peek(tgtRes)
```

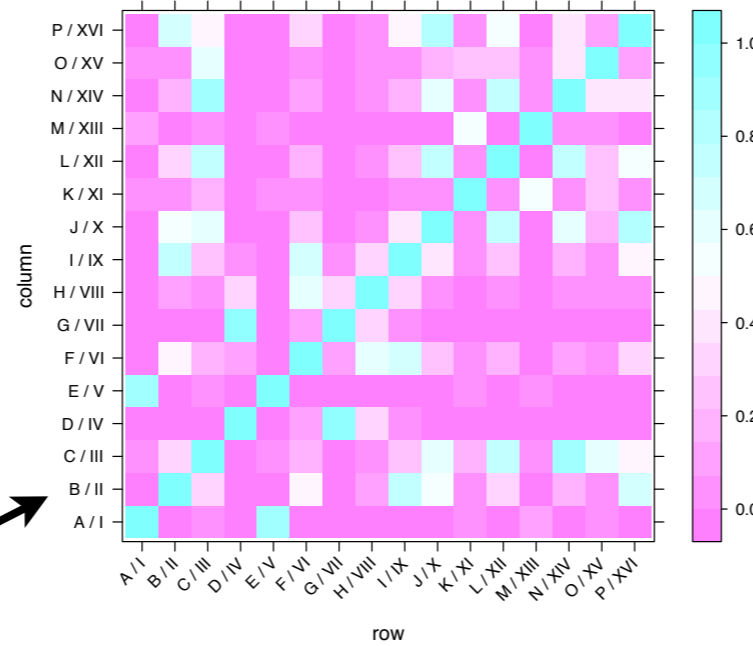
	chromoA	chromoB	t	wilcox	ks	chisq
23	2	10	5.077492e-01	1.260279e-01	4.265804e-02	3.055263e-02
28	2	15	2.749797e-02	1.080776e-04	3.271894e-06	1.146343e-06
43	4	5	7.906832e-22	1.154823e-23	0.000000e+00	4.340871e-19
65	5	16	7.659286e-08	7.746156e-07	7.007464e-06	2.444194e-04
74	6	15	3.601006e-02	2.847043e-03	6.857203e-03	3.464897e-02
90	8	14	1.344403e-02	1.278636e-01	4.284638e-02	5.906893e-03
119	14	16	4.117154e-01	9.092607e-01	2.626741e-01	9.915321e-02

Present 120 p-values in a figure (vs. table)

```

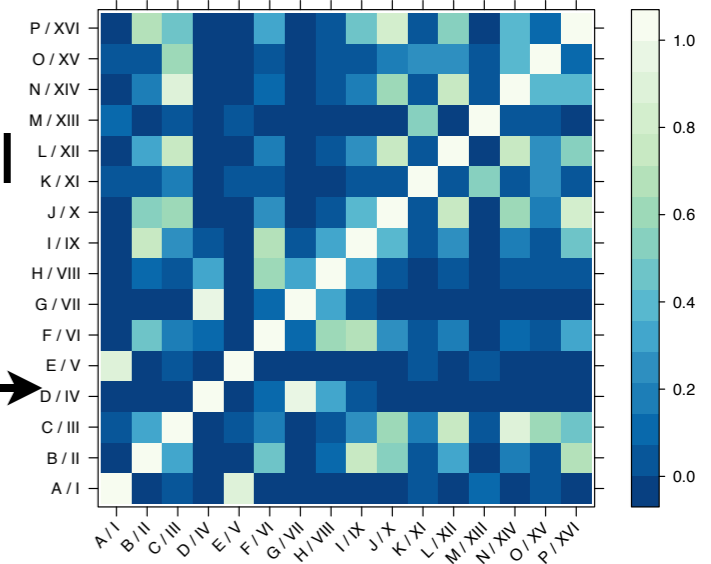
> peek(tgtRes)
  chromoA chromoB      t      wilcox      ks      chisq
23       2      10 5.077492e-01 1.260279e-01 4.265804e-02 3.055263e-02
28       2      15 2.749797e-02 1.080776e-04 3.271894e-06 1.146343e-06
43       4       5 7.906832e-22 1.154823e-23 0.000000e+00 4.340871e-19
65       5      16 7.659286e-08 7.746156e-07 7.007464e-06 2.444194e-04
74       6      15 3.601006e-02 2.847043e-03 6.857203e-03 3.464897e-02
90       8      14 1.344403e-02 1.278636e-01 4.284638e-02 5.906893e-03
119      14      16 4.117154e-01 9.092607e-01 2.626741e-01 9.915321e-02

```



rational color scale

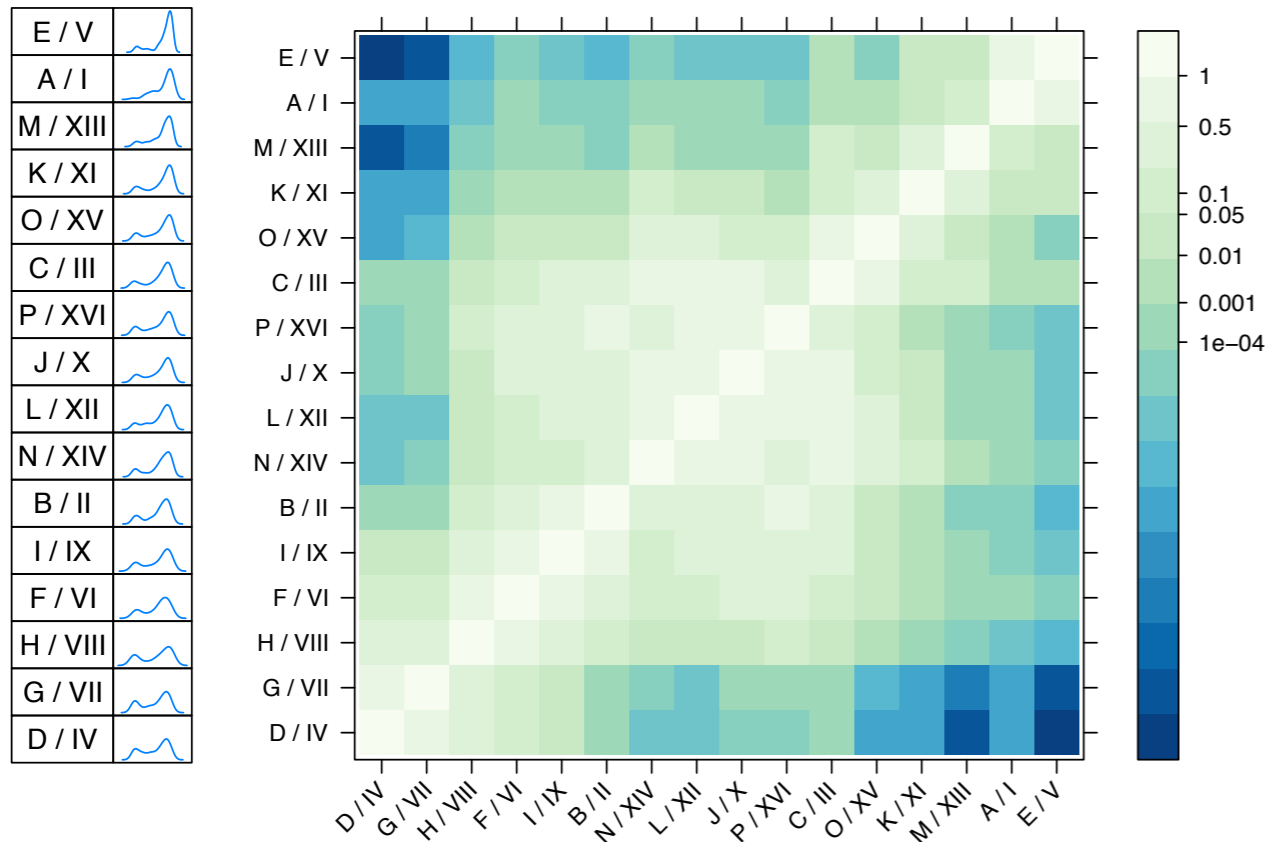
Colors based on 'GnBu' palette from RColorBrewer



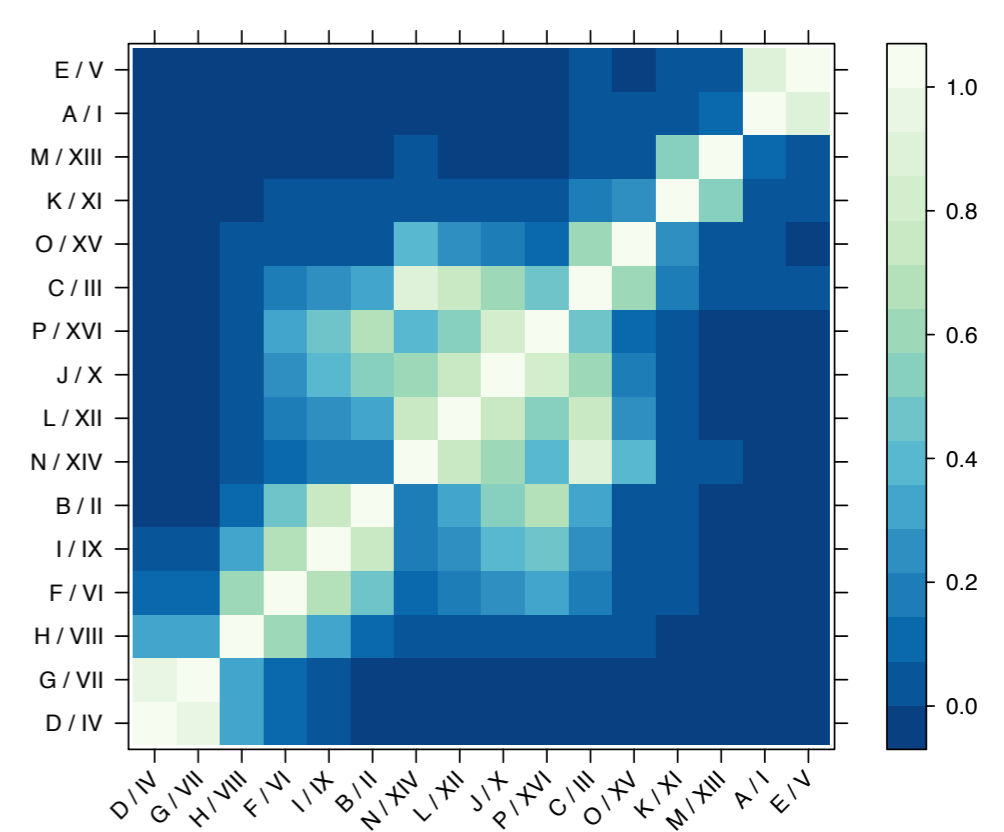
transformation emphasizes important differences

rational order, add densities

Ordered by Jenny, probit transformed p-values



Ordered by Jenny



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 [mylibrary](#)

All of Nonparametric Statistics by Larry Wasserman. Springer, 2006. via [SpringerLink](#) | via [mylibrary](#) | [GoogleBooks](#) search.

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.

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?)

Null hypothesis: $F = G (= H)$

Possible test statistic: $|\text{avg}(x) - \text{avg}(y)|$

Observed value of test statistic = t

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

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 $t^* = |\text{avg } x^* - \text{avg } y^*|$.

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

Compute $t^* = |\text{avg } x^* - \text{avg } y^*|$.

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

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.

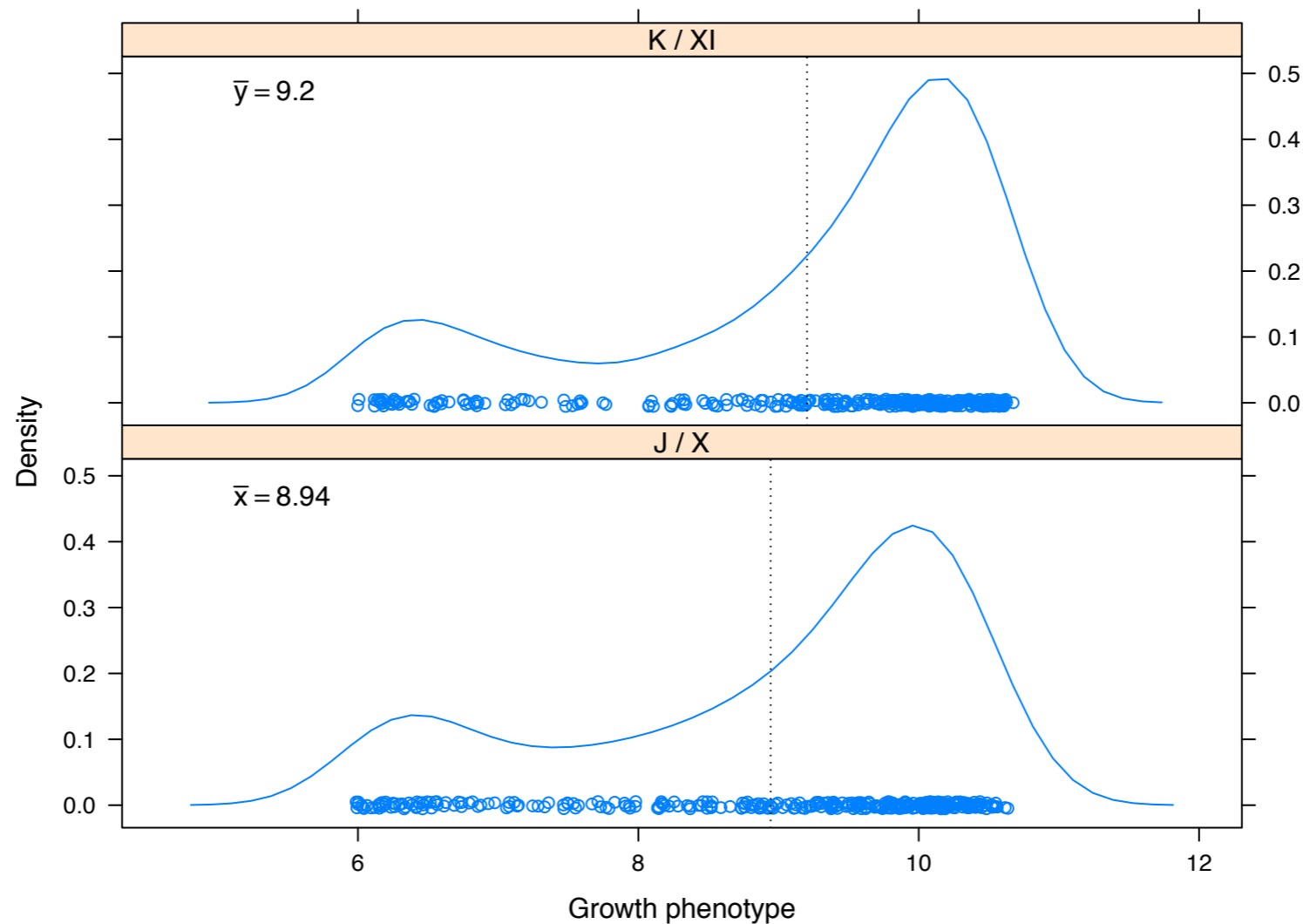
x = data observed from one chromosome, e.g. 10

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

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

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

$F = G?$



Specify a null hypothesis $H_0: F = G (= H)$

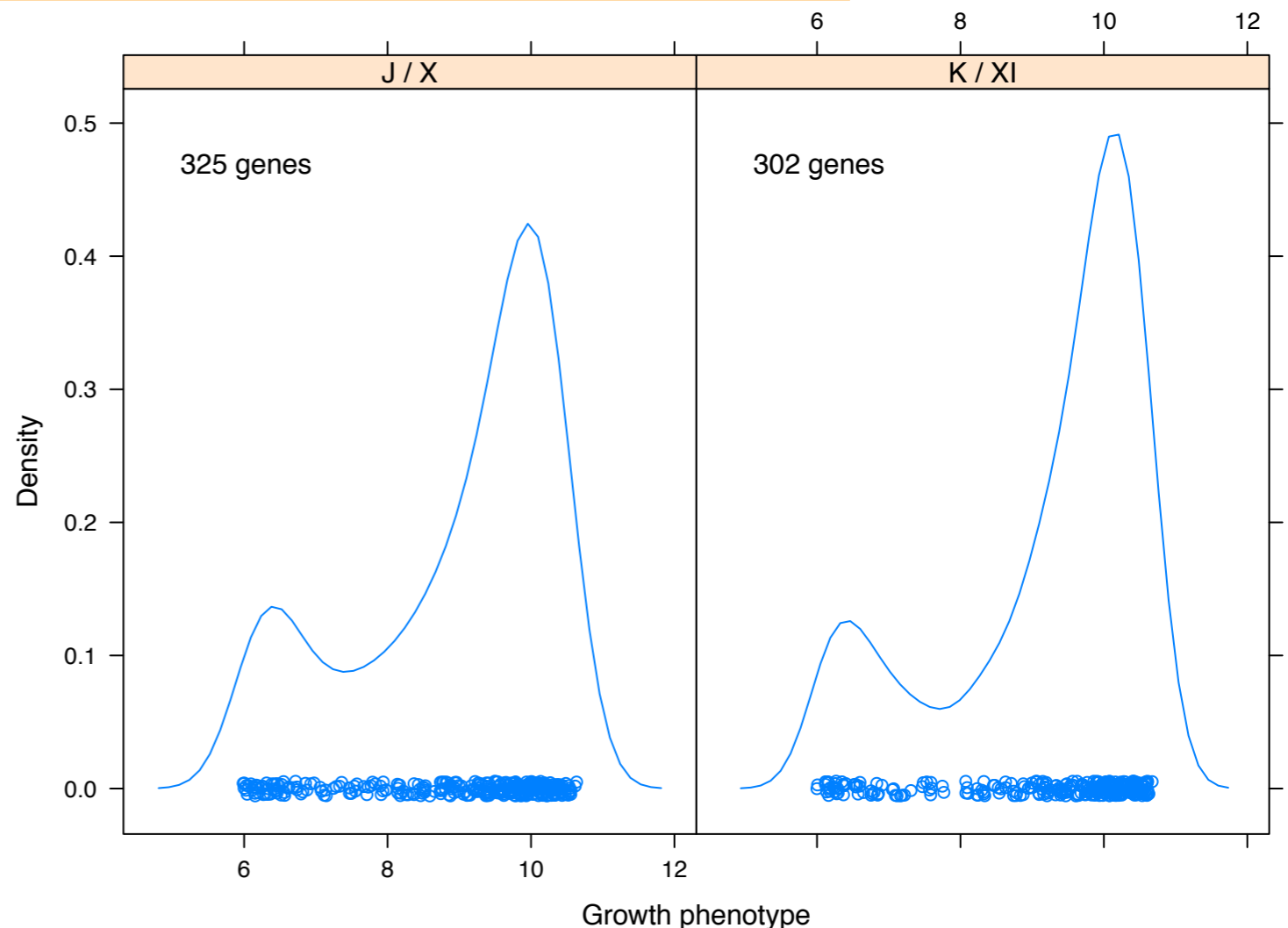
```

jChromo <- c(10, 11)
kDat <- droplevels(subset(hDat, chromo %in% jChromo))
str(kDat) # 627 obs. of 4 variables:
(chromoCounts <- table(kDat$chromo))

densityplot(~ pheno | chromoPretty, kDat,
  xlab = "Growth phenotype",
  panel = function(x, ...) {
    panel.densityplot(x, ...)
    grid.text(paste(length(x), 'genes'),
      x = 0.1, y = 0.9,
      just = c("left", "center"),
      gp = gpar(fontfamily = "sans"))
  })

```

Example of customizing a lattice plot ... re-defining the panel function 'on the fly'. Computing on the 'packet' and writing text on the panel.



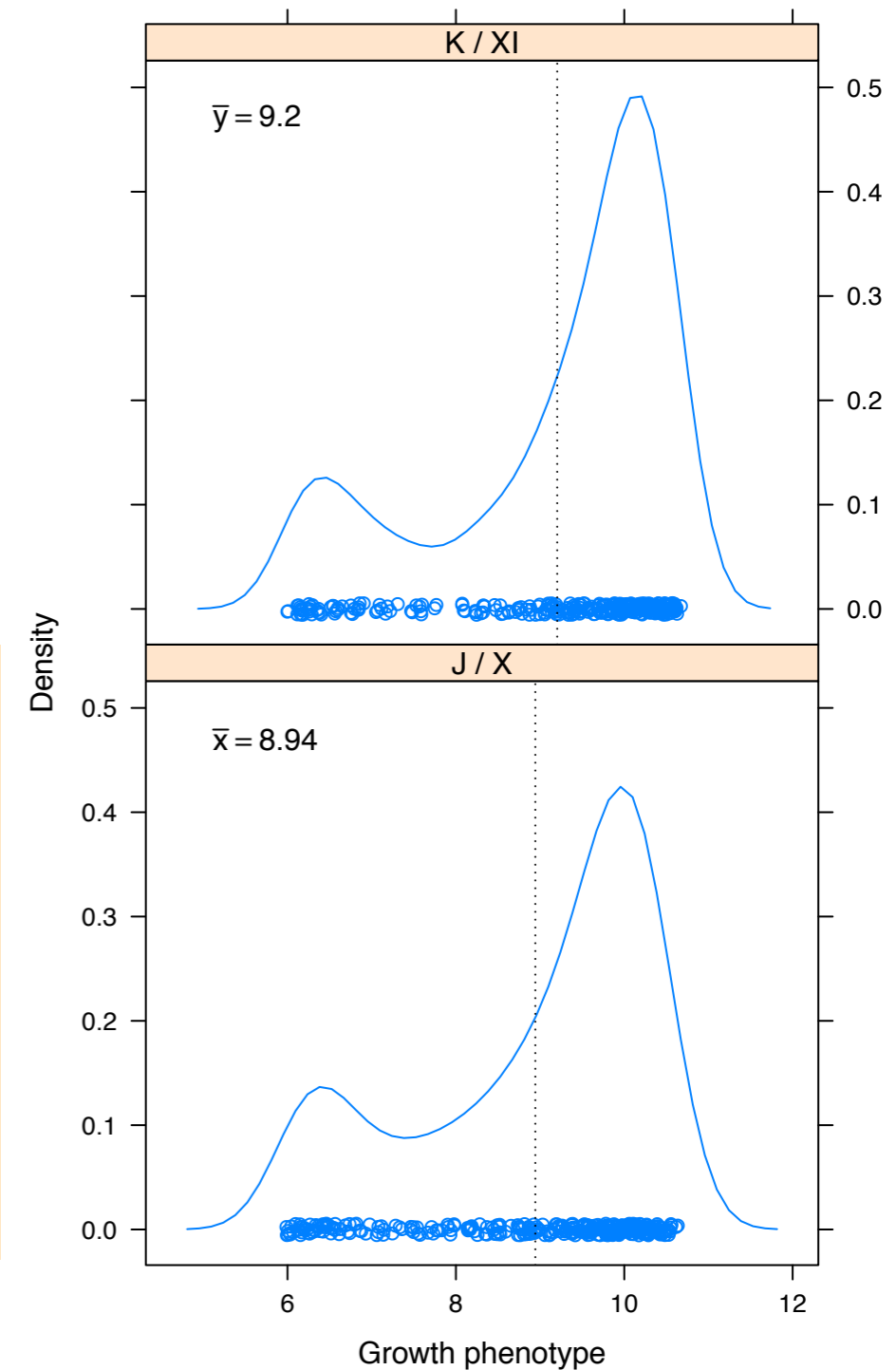
```

> ## isolate data for this pair of chromosomes
> kDat <- droplevels(subset(hDat, chromo %in% jChromo))
...
> ## compute the observed test statistic
> (chromoMeans <- with(kDat,
+                       tapply(pheno, chromo, mean)))
      10      11
8.943558 9.203379

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

> densityplot(~ pheno | chromoPretty, kDat,
+             xlab = "Growth phenotype", layout = c(1,2),
+             panel = function(x, ...) {
+               panel.densityplot(x, ...)
+               mu <- mean(x)
+               panel.abline(v = mu, lty = 'dotted')
+               if(packet.number() == 1) {
+                 avgText <- bquote(bar(x) == .(round(mu, 2)))
+               } else {
+                 avgText <- bquote(bar(y) == .(round(mu, 2)))
+               }
+               grid.text(avgText, x = 0.1, y = 0.9,
+                         just = c("left", "center"))
+             })

```



More computing on the 'packet', adding a vertical line, and writing math on the panel (read up on `plotmath()`).

Choose a test statistic

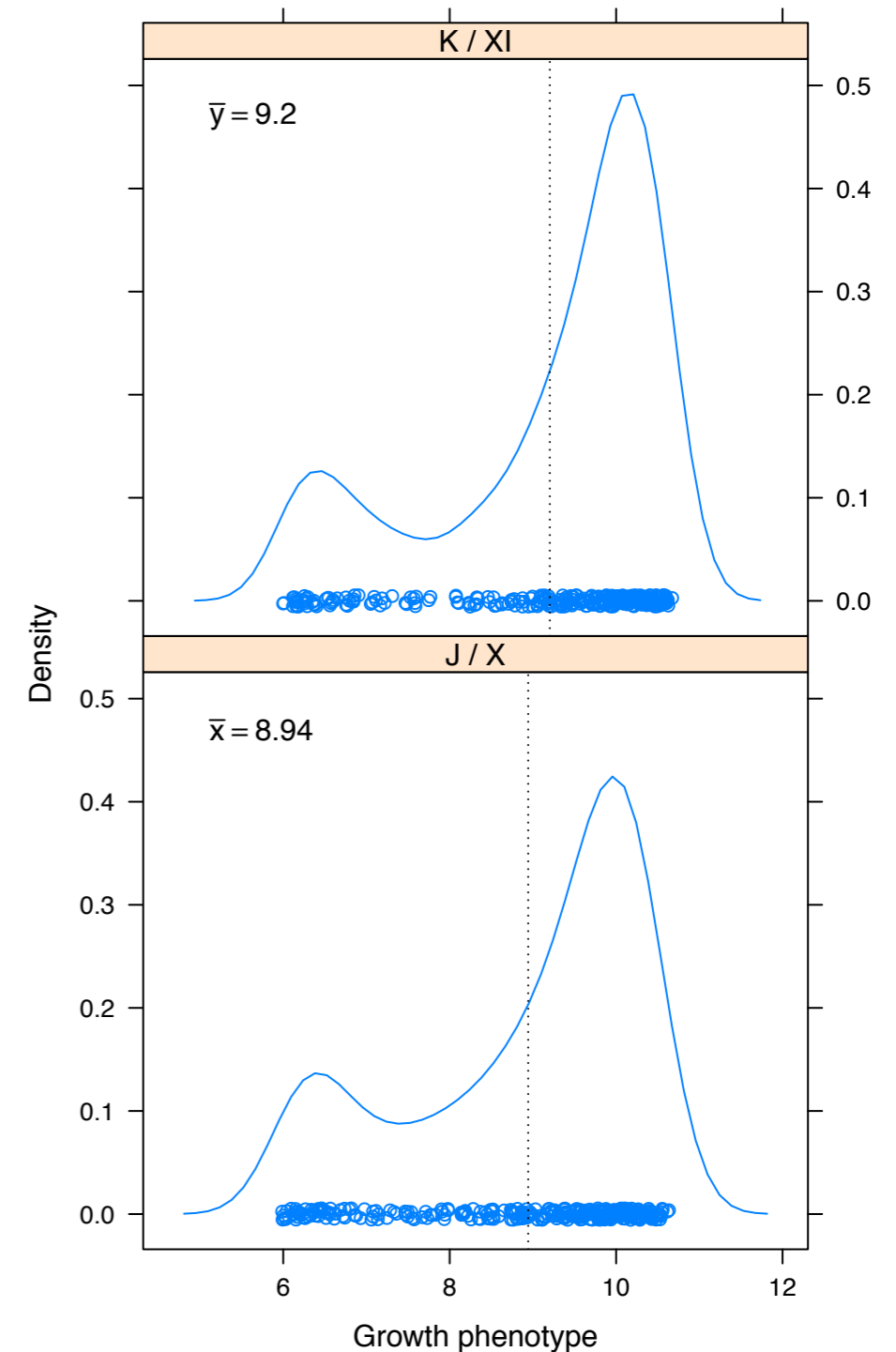
Let's try this: $t = |\bar{x} - \bar{y}|$

```
> (chromoMeans <- with(kDat,  
+                       tapply(pheno, chromo, mean)))  
      10      11  
8.943558 9.203379  
  
> (obsTestStat <- abs(chromoMeans[1] - chromoMeans[2]))  
      10  
0.2598215
```

$$\bar{x} = 8.94$$

$$\bar{y} = 9.2$$

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



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

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

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
```

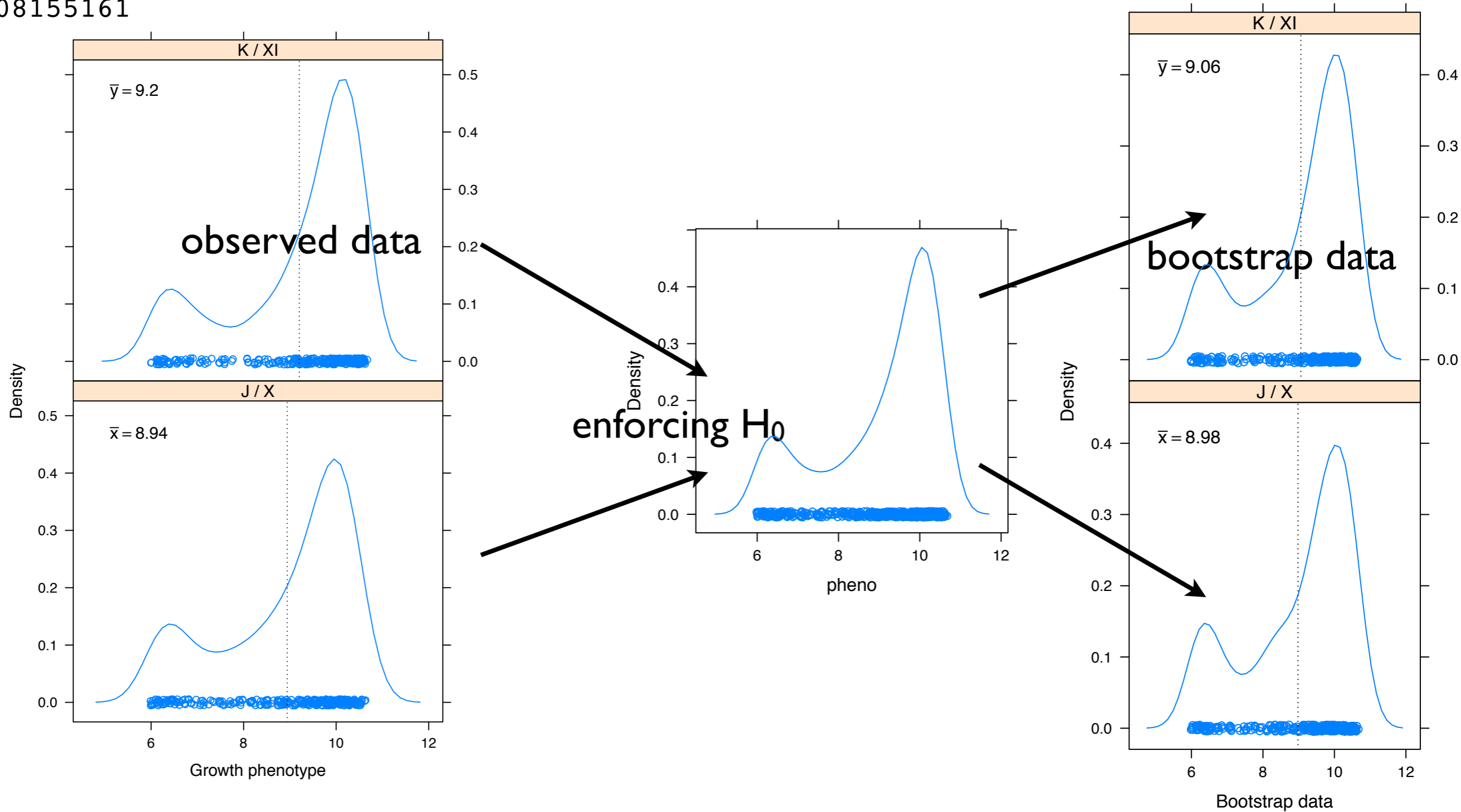
Generate bootstrap data by resampling with replacement. Draw n_x observations from it and call this x^* , generate another n_y observations and call this y^* .

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

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

<snip, snip>

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



Observed value of test statistic = t

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

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

Compute $t^* = |\text{avg } x^* - \text{avg } y^*|$ from a bootstrap sample.

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

```
> z <- kDat$pheno
> xStar <- sample(z, size = chromoCounts[1], replace = TRUE)
> yStar <- sample(z, size = chromoCounts[2], replace = TRUE)
> (bootTestStat <- abs(mean(xStar) - mean(yStar)))
[1] 0.08155161
```

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

```

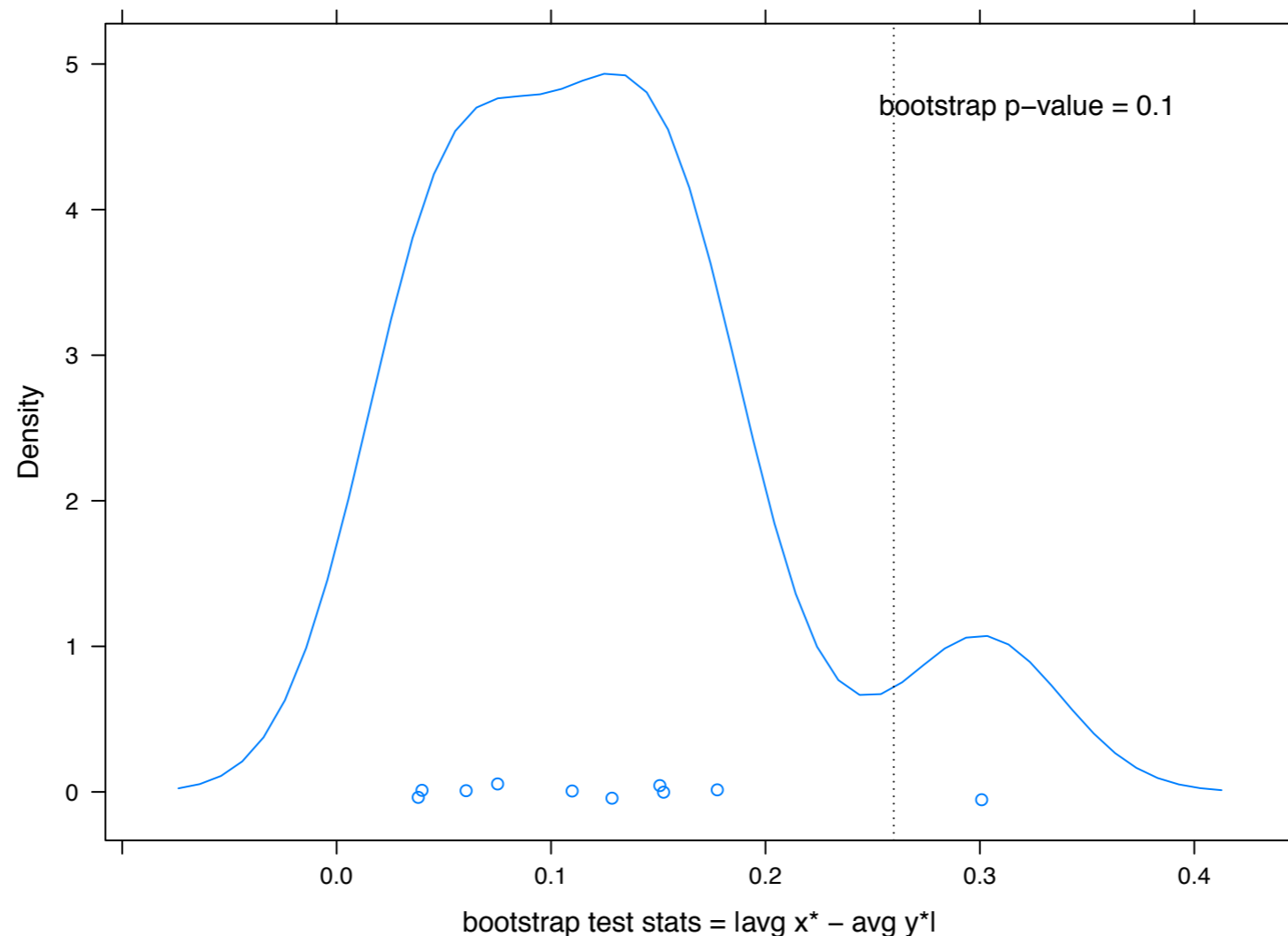
What proportion of the t^* are as or more extreme as t ? That's basically your bootstrap p-value.

```
> (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.

How to ...

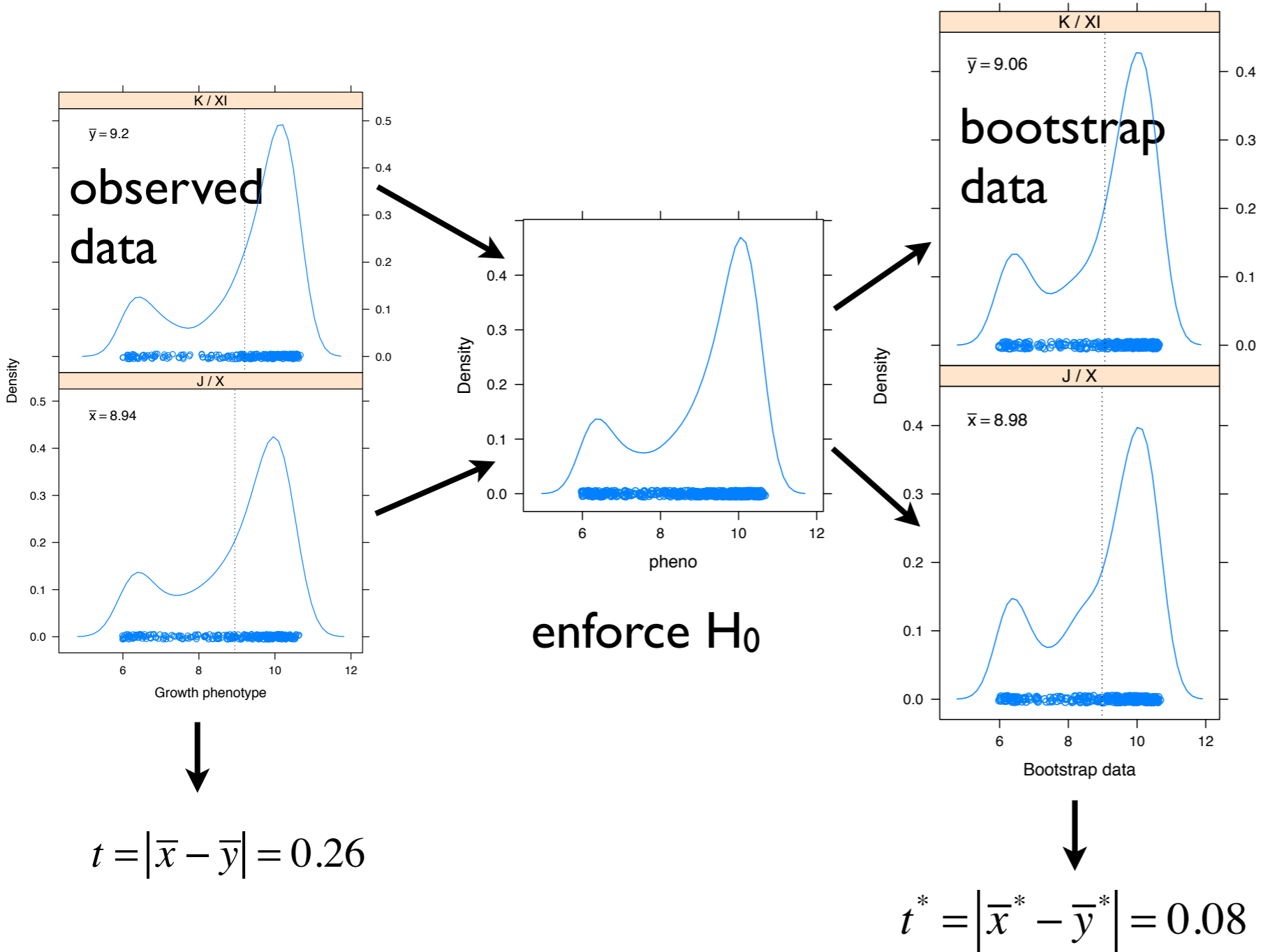
Determine the distribution for the test statistic under H_0

???

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.

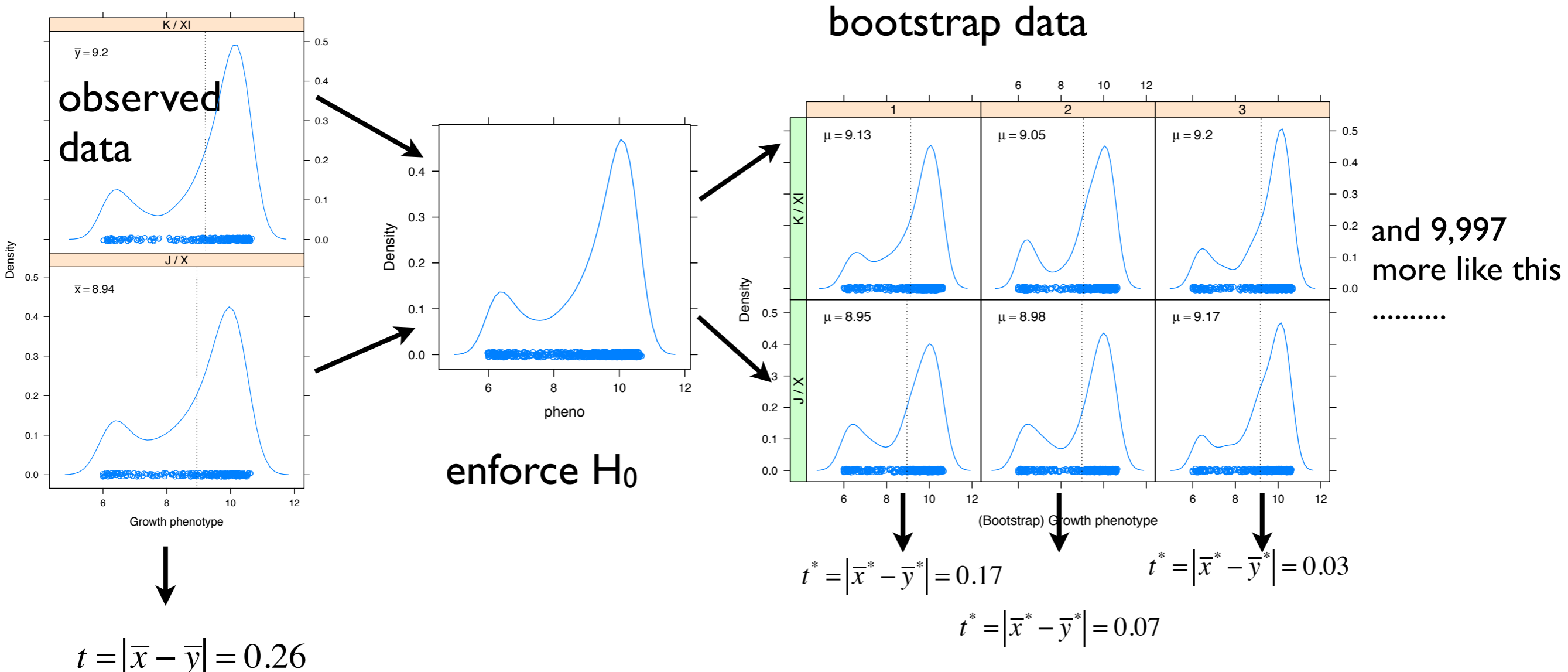


```

> z <- kDat$pheno
> xStar <- sample(z, size = chromoCounts[1], replace = TRUE)
> yStar <- sample(z, size = chromoCounts[2], replace = TRUE)
> abs(mean(xStar) - mean(yStar)) # bootstrap test statistic
[1] 0.08155161

```

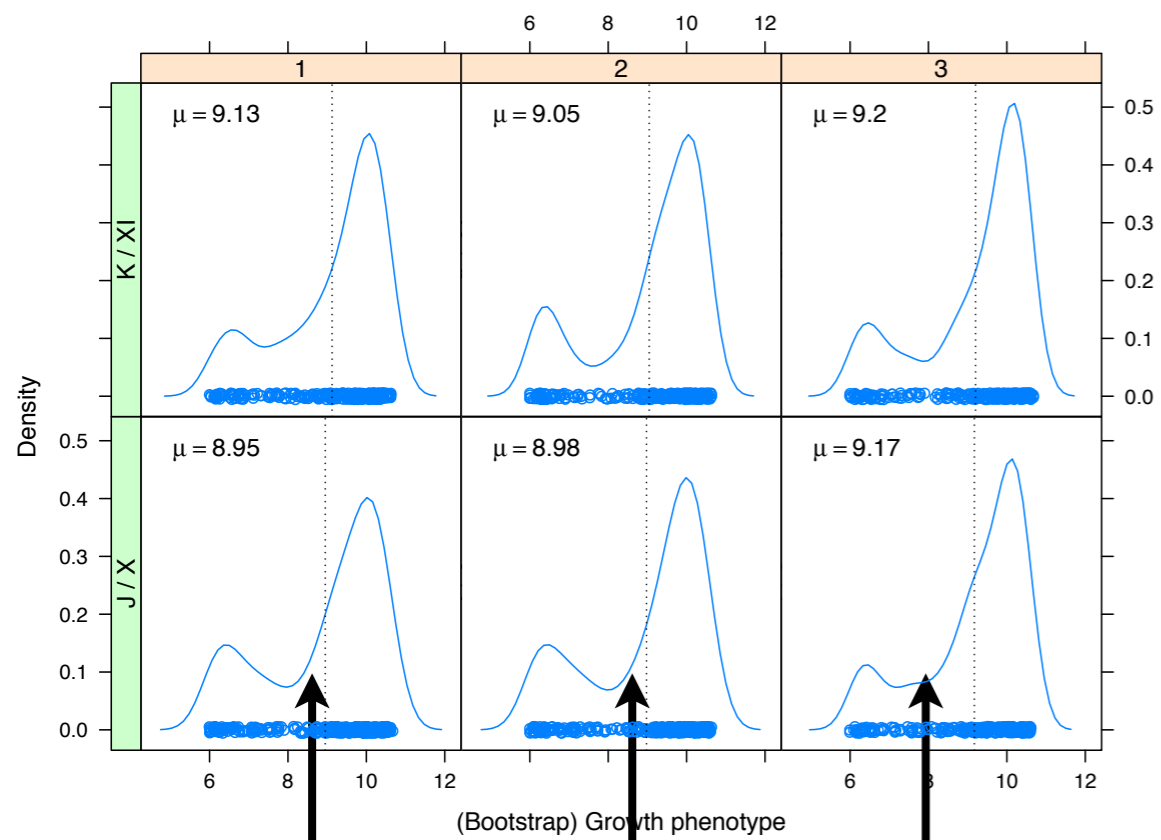
I bootstrap sample (“baby steps”)



```
(n <- nrow(kDat))
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)
```

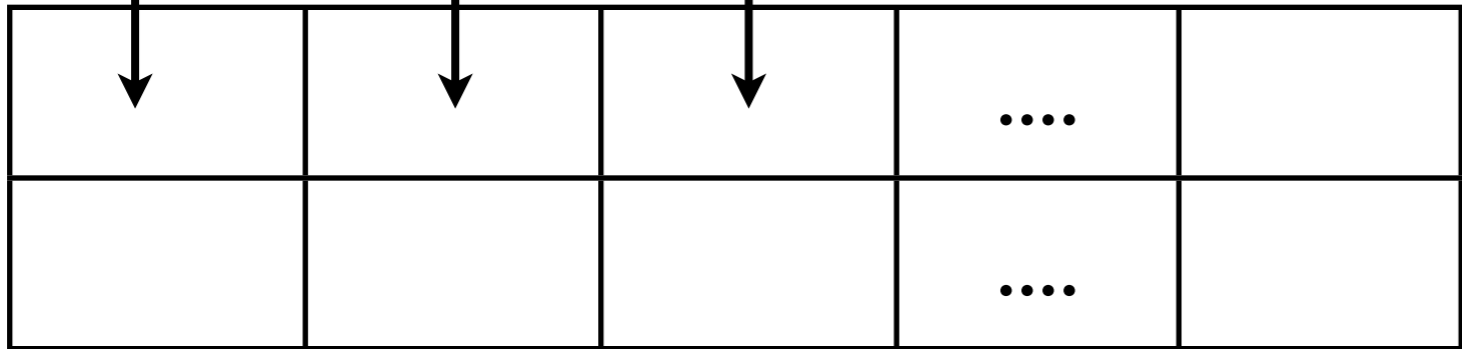
No explicit loops!

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



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

each column is 1 bootstrap sample

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')
+   })

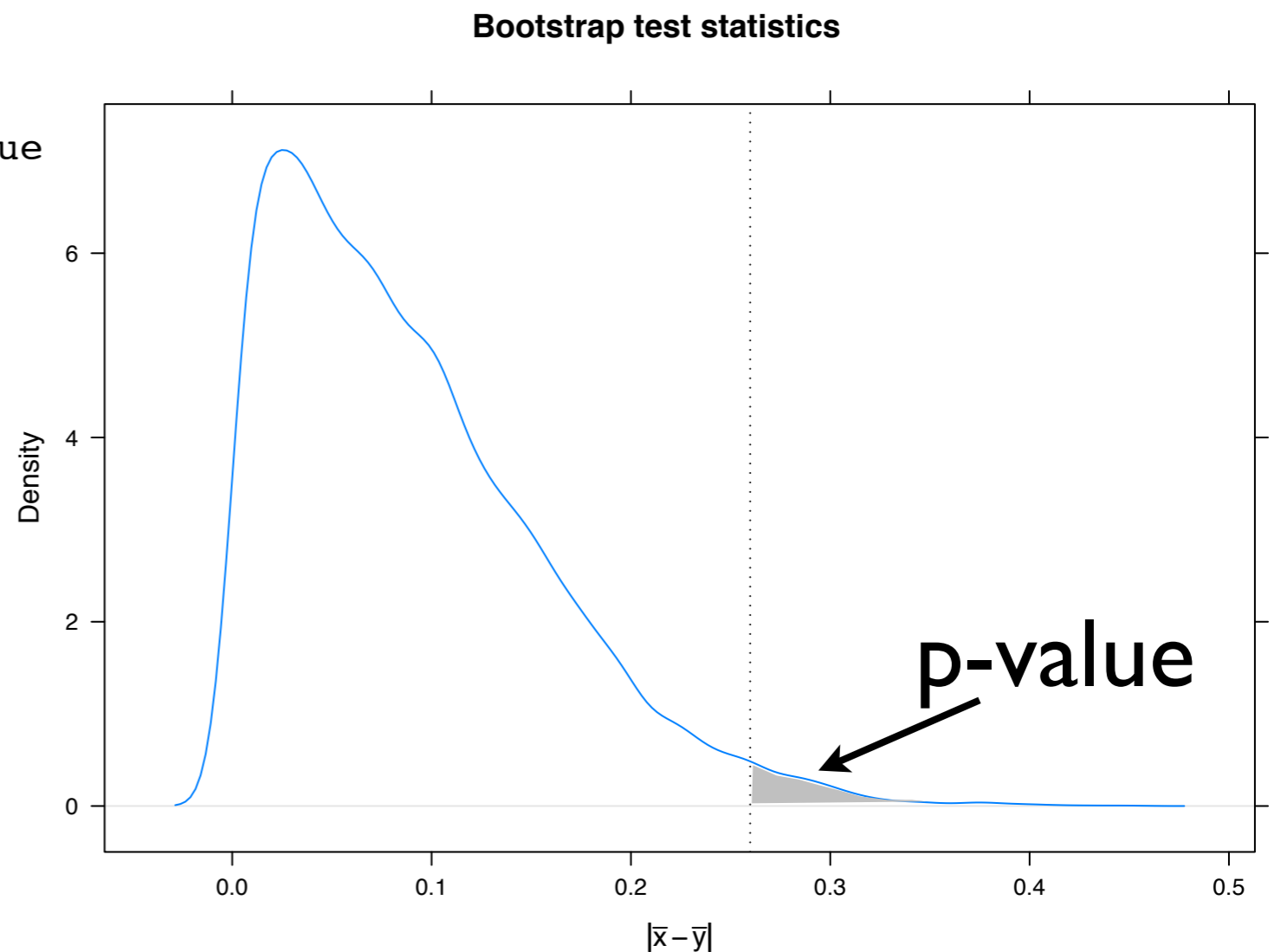
> ## bootstrap p-value
> mean(bootTestStats >= obsTestStat)
[1] 0.0172

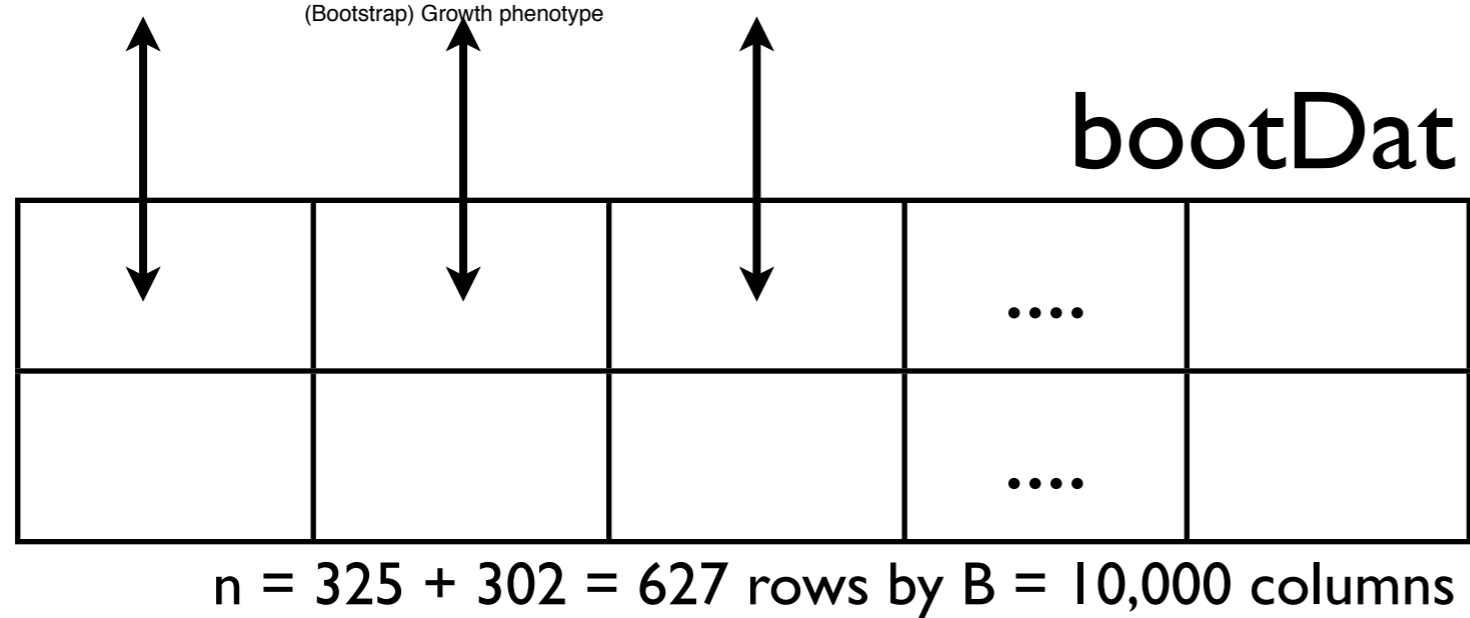
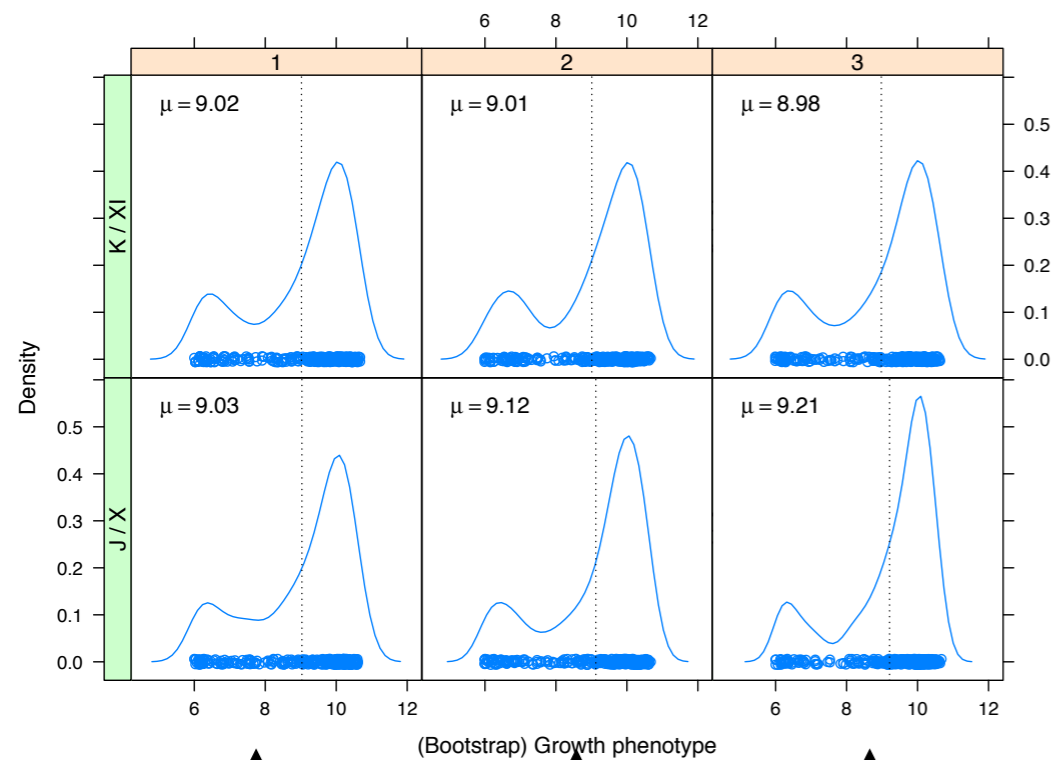
> t.test(pheno ~ chromo, kDat)$p.value
[1] 0.01940612

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

```

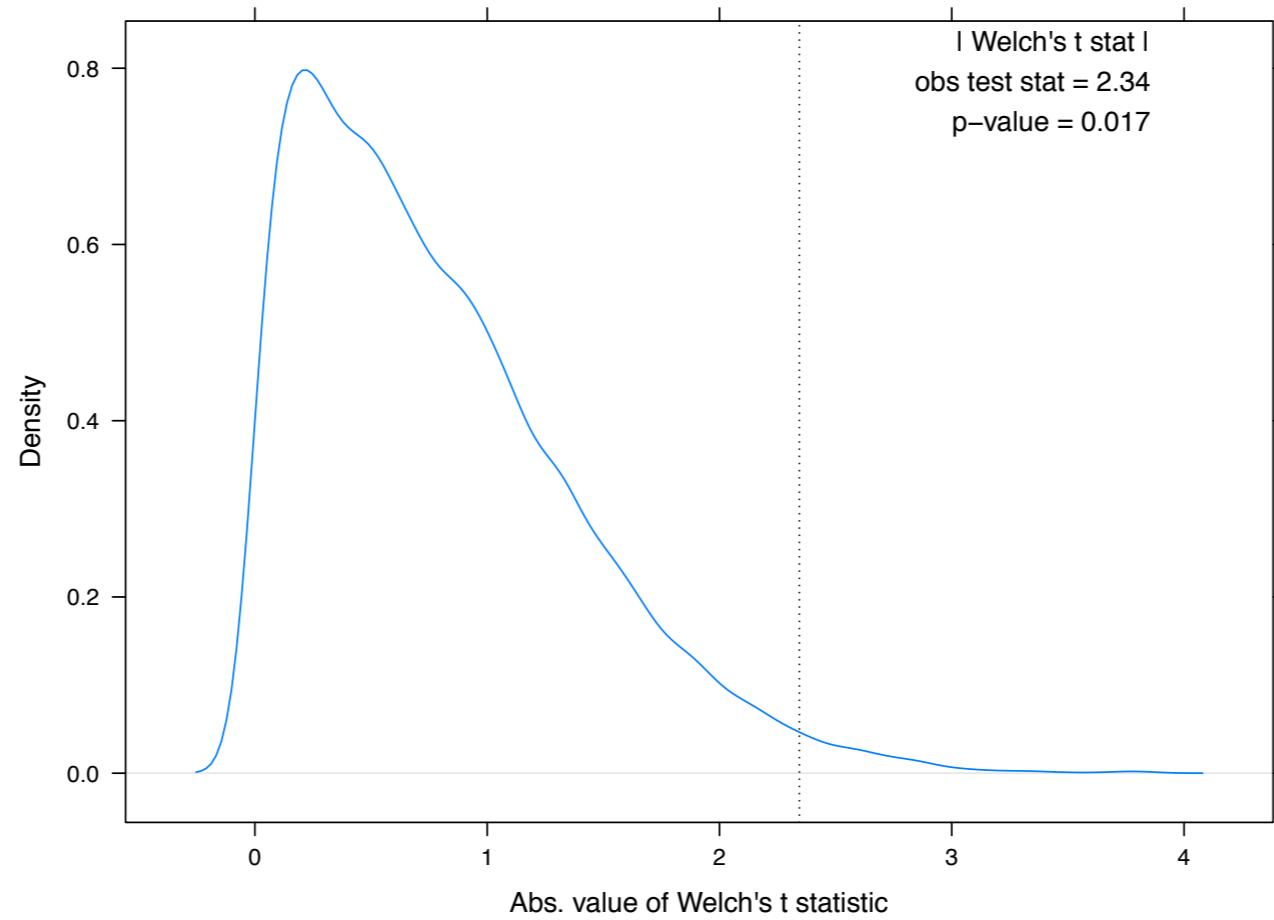
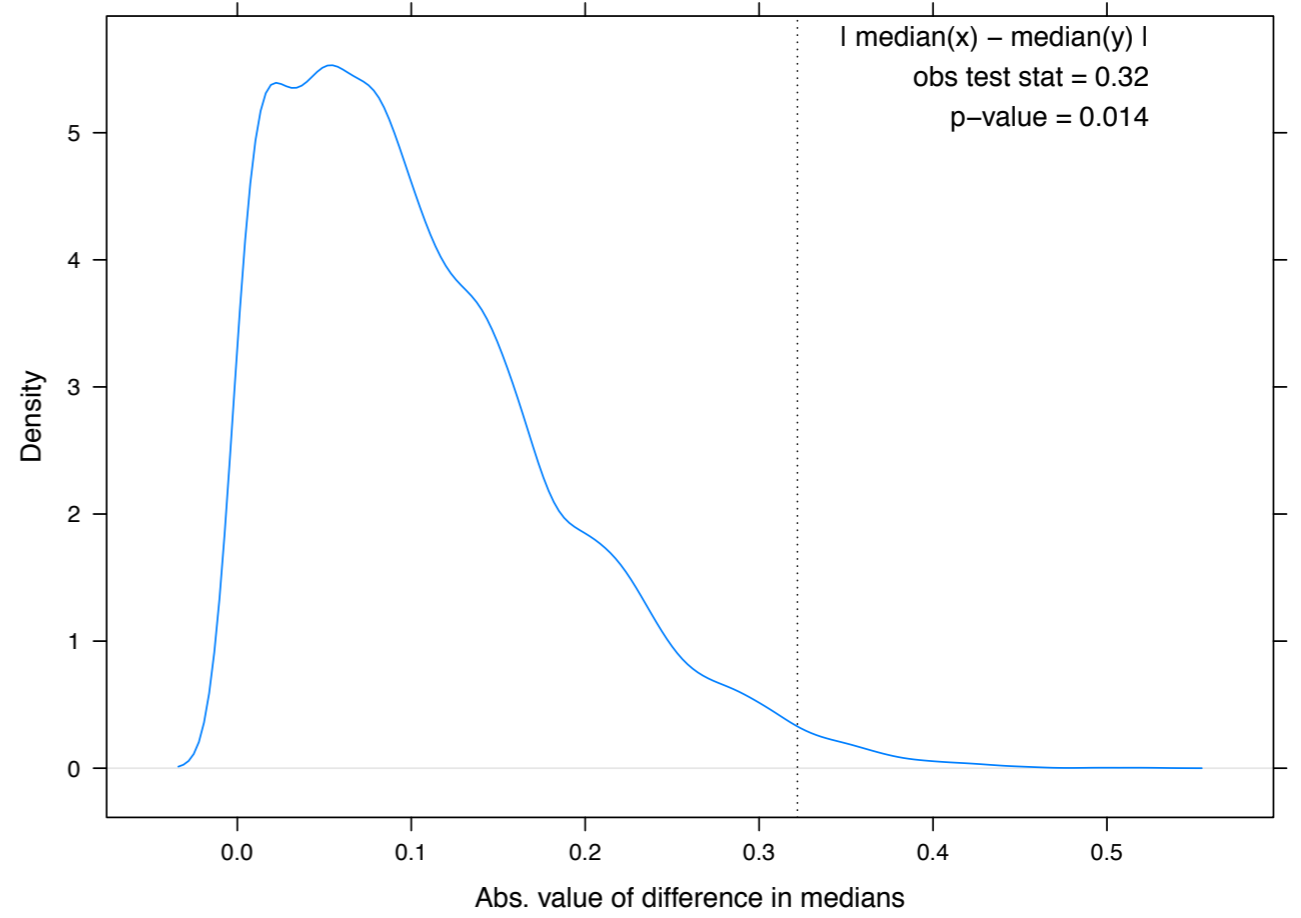
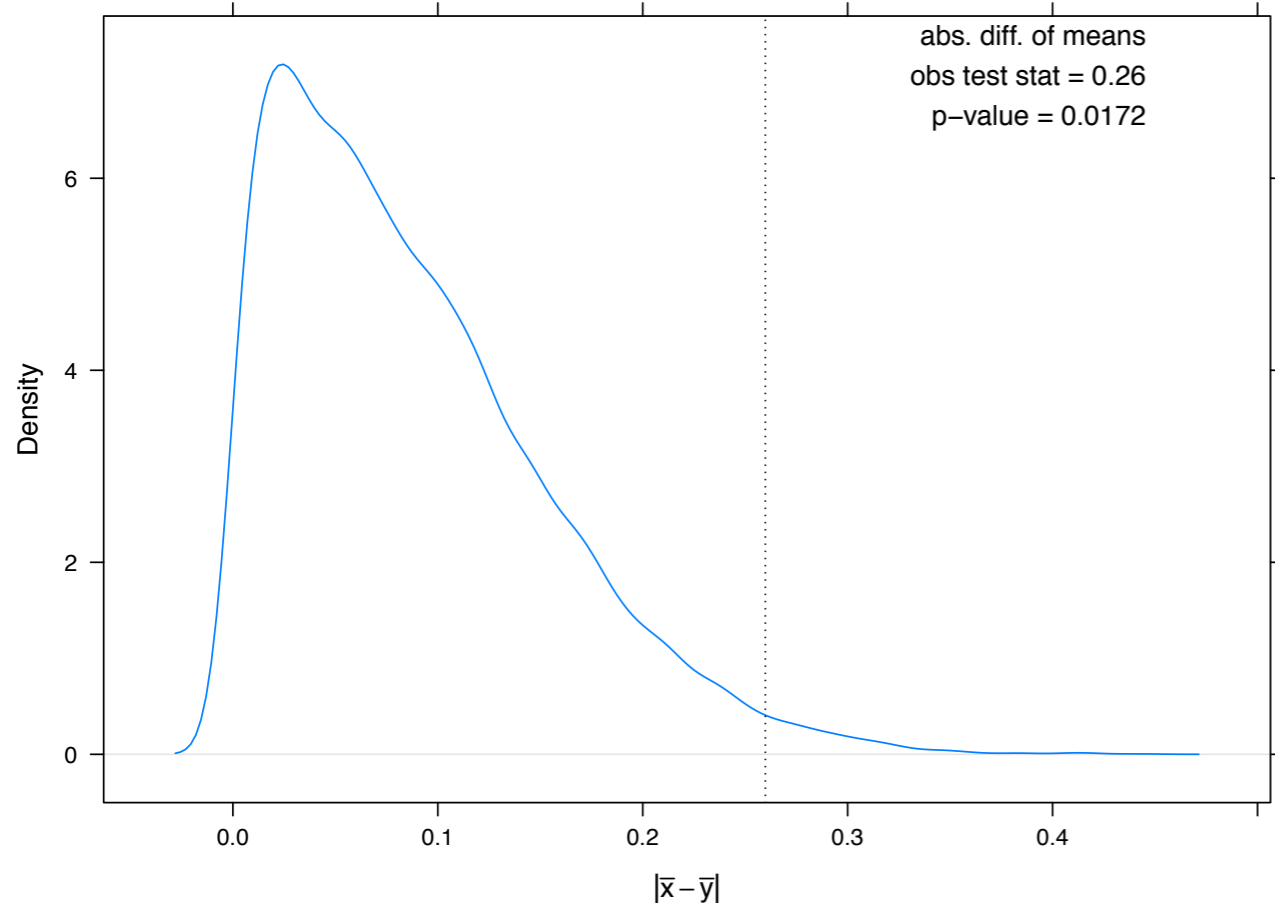
Bootstrap p-value is awfully close to that from Welch's t-test. That's comforting. Wilcoxon p-value is yet smaller -- perhaps due to greater efficiency in large samples w/ non-normal data.

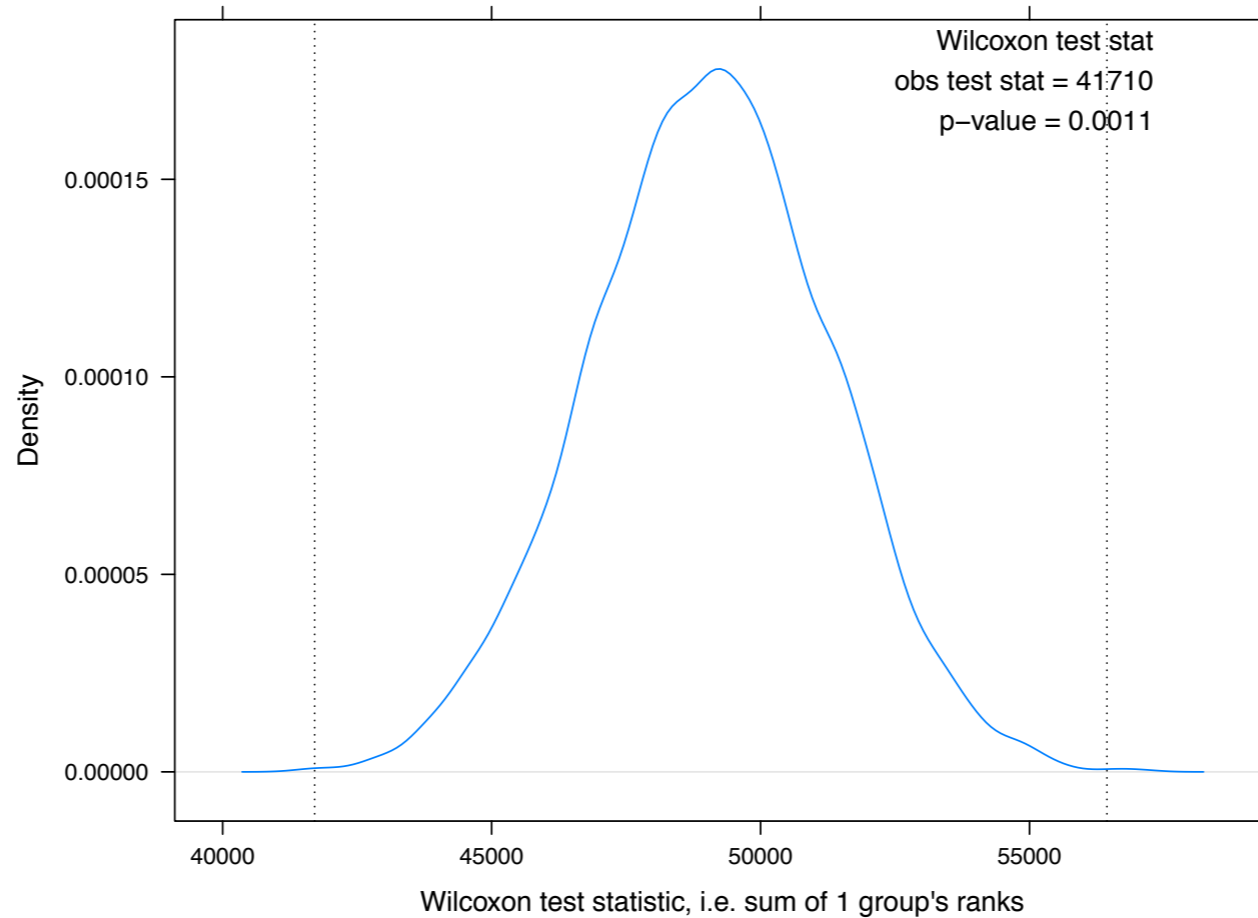
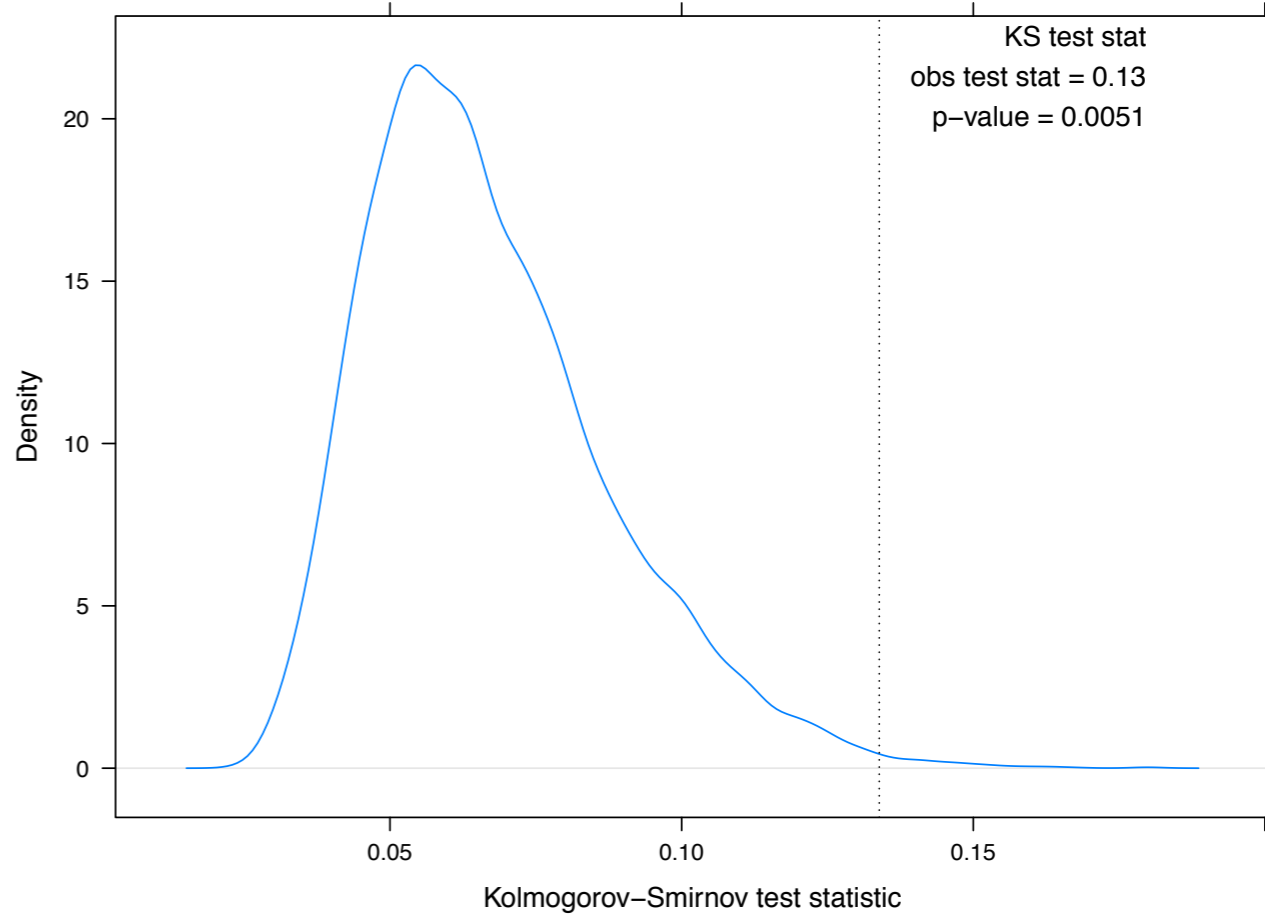
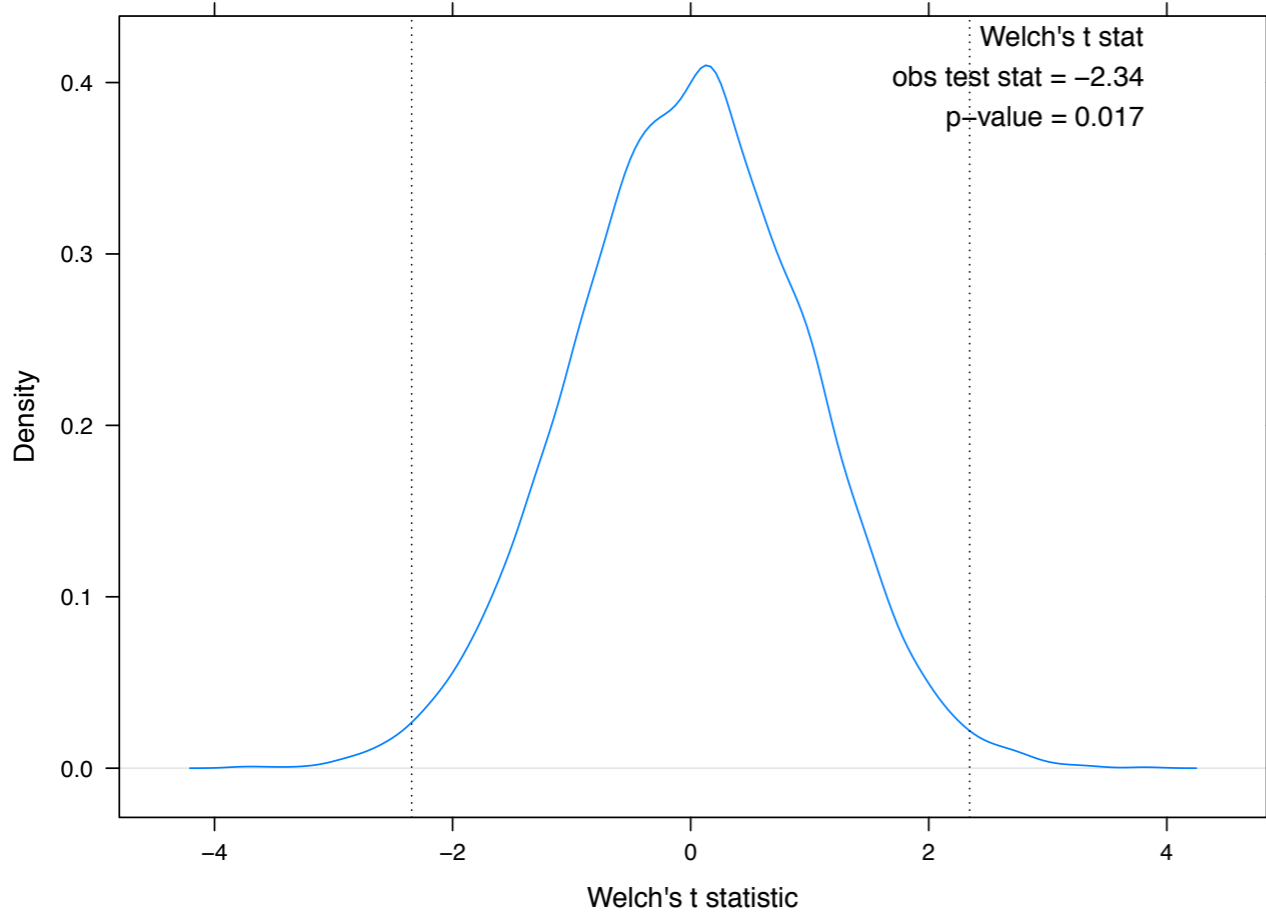




each column is 1
bootstrap sample

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.





test statistic	observed value	“classical” p-value	bootstrap p-value
$t = \bar{x} - \bar{y} $	0.26	NA	0.0172
$t = \text{median}(x) - \text{median}(y) $	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 $\langle \text{sthgSimple} \rangle$ or a distribution less tractable than $\langle \text{sthgFriendly} \rangle$, 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.”

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.”

those wise words apply to many “brute force” methods of statistical inference and analysis

on that note, let’s return to the world of two quantitative variables and take a tour of some nonstandard regression approaches:

today -- robust regression (and the bootstrap, again!)

Wednesday -- smoothing

[CRAN Task View: Robust Statistical Methods](#)

JB kindly thanks Matias Salibian-Barrera for offering advice on this module several years ago; any mistakes, however, are completely mine!

Please note that there is a package to help with bootstrapping: `boot`.

It is a recommended package and, therefore, will already be in most R installations.

Feel free to try it out.

as usual, code, figures, etc. can be found here:
[robustRegression](#)

Two quantitative variables: X and Y

- X ... ‘independent variable’, ‘covariate’, ‘predictor’, ‘explanatory variable’
- Y ... ‘dependent variable’, ‘response’, ‘outcome’
- Regression \approx study of the conditional expectation of Y given $X=x$

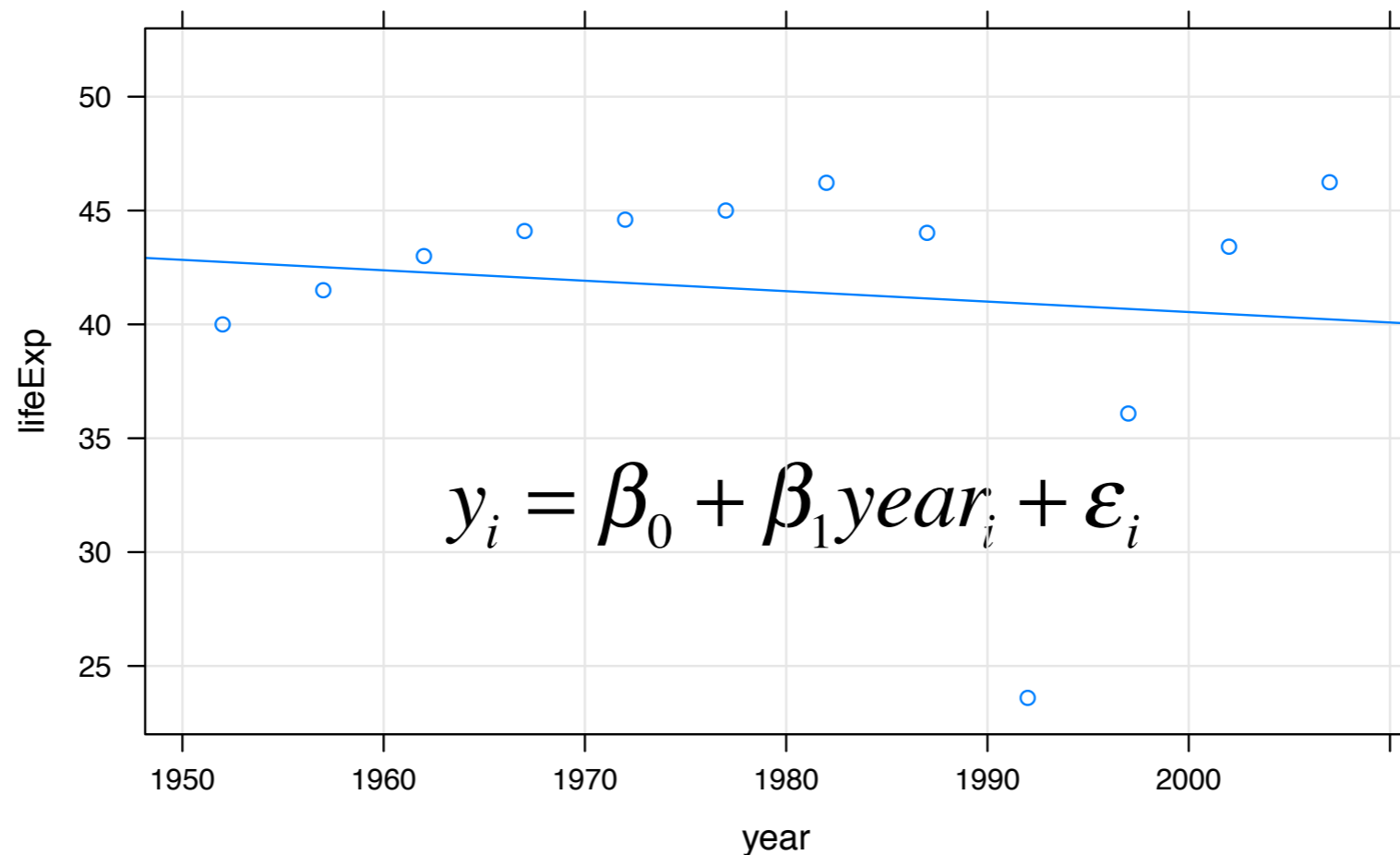
$$y_i = f(x_i) + \varepsilon_i$$

- Our focus: We’re willing to specify quite a bit about f -- maybe even that it’s linear! -- but we’re worried that the ε ’s are a mix of “well-behaved” errors (e.g. mean zero, finite variance) and some really nasty stuff.

Wikipedia: “Rwandans form three groups: the Hutu, Tutsi, and Twa.... The Tutsi-led Rwandan Patriotic Front (RPF) launched a civil war in 1990, which was followed by the 1994 Genocide, in which Hutu extremists killed an estimated 500,000 to 1 million Tutsi and moderate Hutu. The RPF ended the genocide with a military victory.” (Estimated 2011 population = 11.4 million)

Rwanda



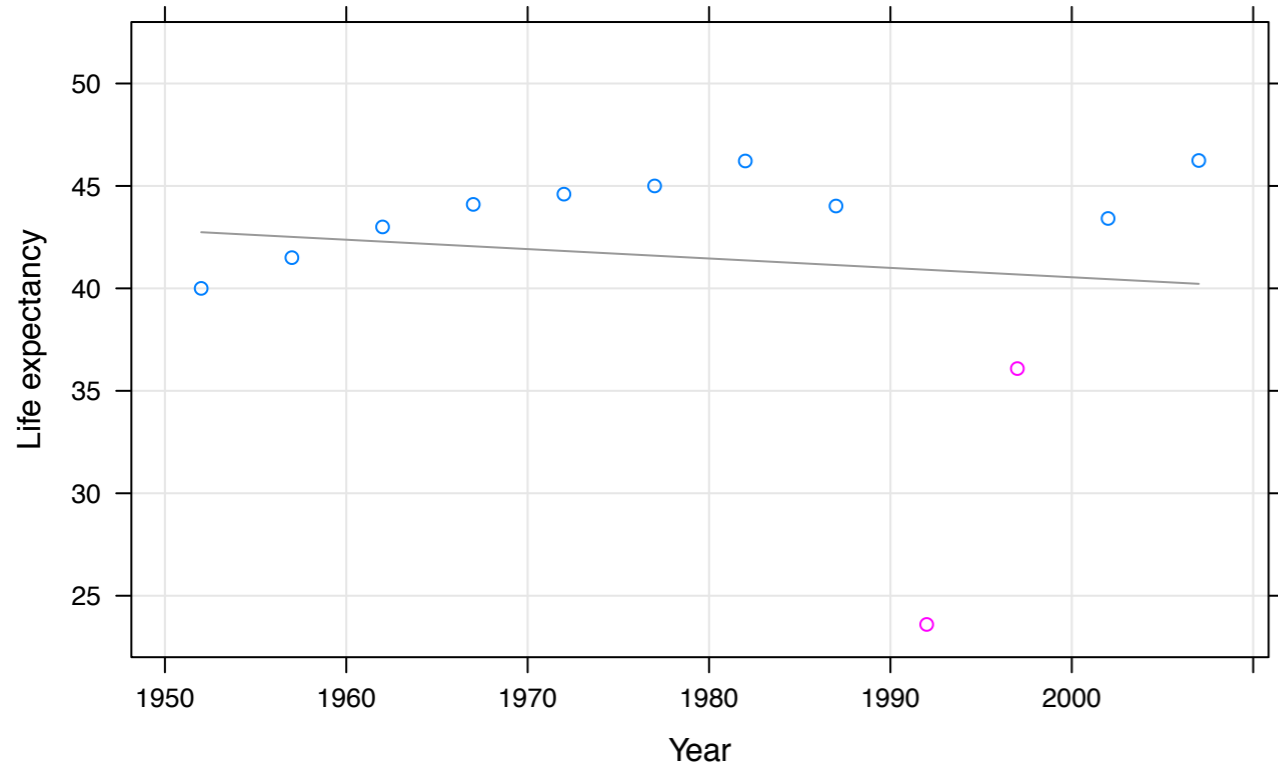


Is life expectancy really declining in Rwanda, in a systematic long-term way?

Or is the above simple linear regression line being dragged down by the two “outliers” heavily affected by the civil war and genocide?

Let's try some robust approaches to regression.

* In real life, I would address the correlation structure of the errors! Ignoring for simplicity!

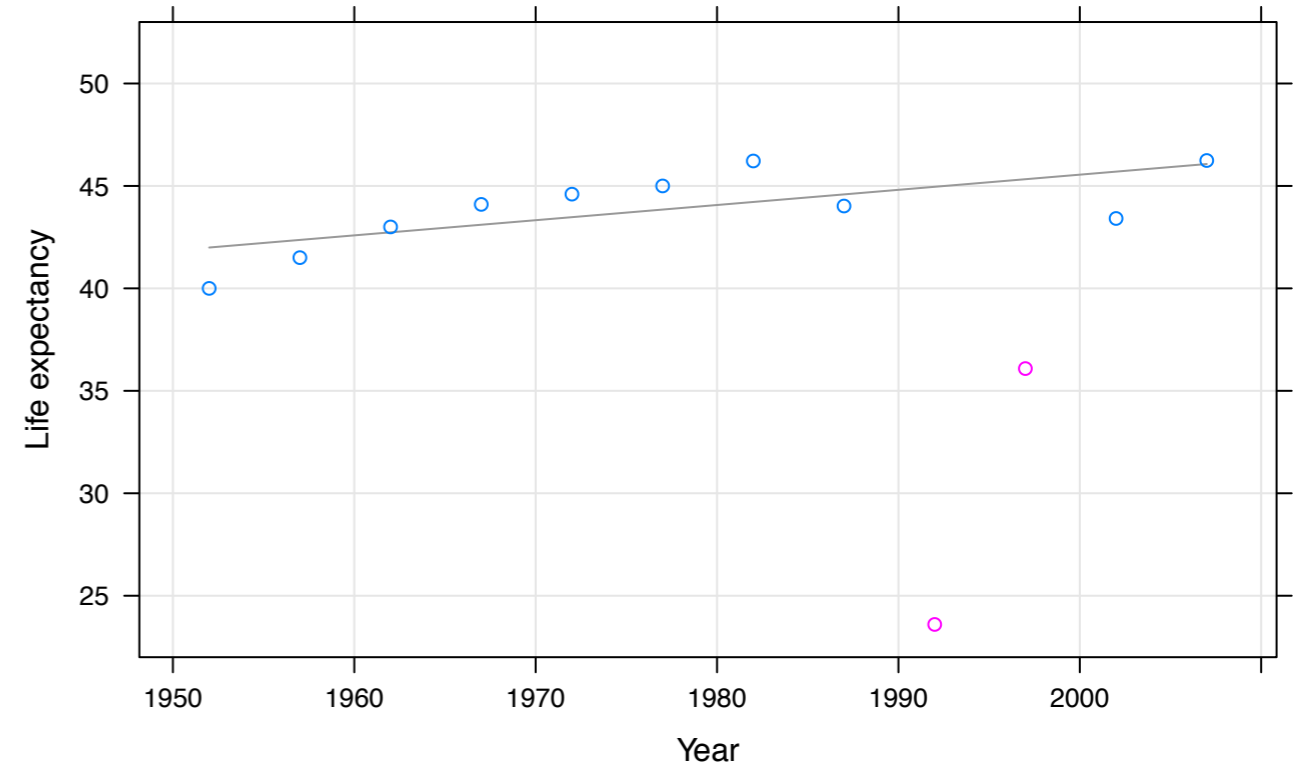


```
> summary(lsFit)
```

```
Call:
lm(formula = lifeExp ~ I(year - yearMin), data = hDat)
```

```
Residuals:
    Min     1Q   Median     3Q     Max
-17.310  -1.445   2.410   3.073   6.021
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    42.74195     3.56128  12.002 2.92e-07 ***
I(year - yearMin) -0.04583     0.10969  -0.418  0.685
```



```
> summary(lsFitAlt)
```

```
Call:
lm(formula = lifeExp ~ I(year - yearMin), data = hDat,
    subset = flag == "data OK")
```

```
Residuals:
    Min     1Q   Median     3Q     Max
-2.2855 -0.7903  0.2190  1.0917  2.0012
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    41.99422     0.81870  51.294 2.31e-11 ***
I(year - yearMin)  0.07408     0.02725   2.718  0.0263 *
```

Robust or resistant regression

- Imagine a scenario where you have real concern about the undue influence of a handful of observations
- If you think of `lm` as implementing least squares regression, then ... perhaps we should consider alternative measures of discrepancy between observed data and the prediction or fit from a model.
- Goal: Devise a fitting criterion that is less sensitive to individual observations.

“residual” $\varepsilon_i(\beta) = y_i - x_i\beta$

default estimator of β

$$\hat{\beta} = (X^T X)^{-1} X^T y = \min^{-1} \sum (y_i - x_i\beta)^2$$

default rationale:

find value of β that minimizes the sum or --
equivalently -- the mean of the squared residuals

If you're worried about undue influence of some outliers, you could replace the mean with a more robust measure of location.

For example, find value of β that minimizes the median of the squared residuals.

Or, find value of β that minimizes the trimmed mean of the squared residuals.

We observe (y_i, x_i) , where x_i is the p -dimensional predictor (a row vector). Considering the data fixed, the residuals are a function of the parameter β :

$$\varepsilon_i(\beta) = y_i - x_i\beta$$

The ‘usual’ estimator of β , i.e. that implemented by `lm`, is justified either as a maximum likelihood estimator (normally distributed errors) or simply via least squares:

$$\hat{\beta} = (X^T X)^{-1} X^T y = \min^{-1} \sum (y_i - x_i\beta)^2$$

So, obviously, there is a great deal of theory and history behind the idea of minimizing the sum or, equivalently, the mean of the squared residuals.

$$\hat{\beta}_{LS} = \min^{-1} \frac{1}{n} \sum (y_i - x_i \beta)^2 = \min^{-1} \text{mean}(\varepsilon^2_i(\beta))$$

Since the median is a robust alternative to the mean, maybe we should consider an estimator that minimizes the median of the squared residuals.

$$\hat{\beta}_{LMS} = \min^{-1} \text{median}(\varepsilon^2_i(\beta))$$

This is called the least median of squares (LMS) estimator.

You could imagine replacing the ‘mean’ or ‘median’ above with ... a trimmed mean. That leads to the least trimmed squares (LTS) regression.

Let's replace the 'mean' with a 'trimmed mean'. That leads to least trimmed squares (LTS) regression.

$$\hat{\beta}_{LTS} = \min^{-1} \text{tr mean}_{\tau}(\varepsilon^2_i(\beta))$$

where $\tau \in (0, 1)$ specifies the amount of trimming.

As with most tuning parameters, there is some information on an optimal choice of τ and well-designed software should default to that.

$$\hat{\beta}_{LS} = \min^{-1} \frac{1}{n} \sum (y_i - x_i \beta)^2 = \min^{-1} \text{mean}(\varepsilon^2_i(\beta))$$

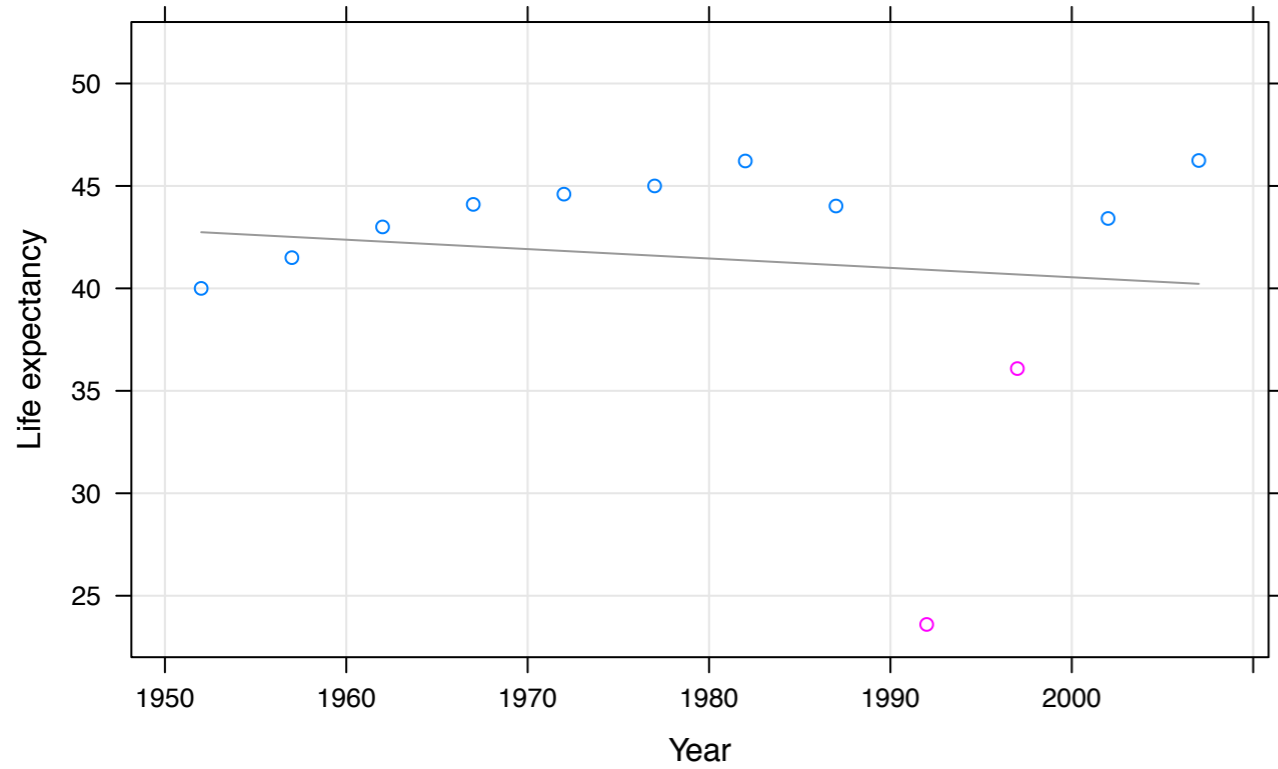
~~$$\hat{\beta}_{LMS} = \min^{-1} \text{median}(\varepsilon^2_i(\beta))$$~~

removed from comparison because other robust options are superior

$$\hat{\beta}_{LTS} = \min^{-1} \text{tr mean}_{\tau}(\varepsilon^2_i(\beta))$$

$\hat{\beta}_{MM}$ state-of-the-art robust estimator, but not easy to explain; gold standard

Can the robust estimators achieve this but without the need to explicitly, subjectively remove data?

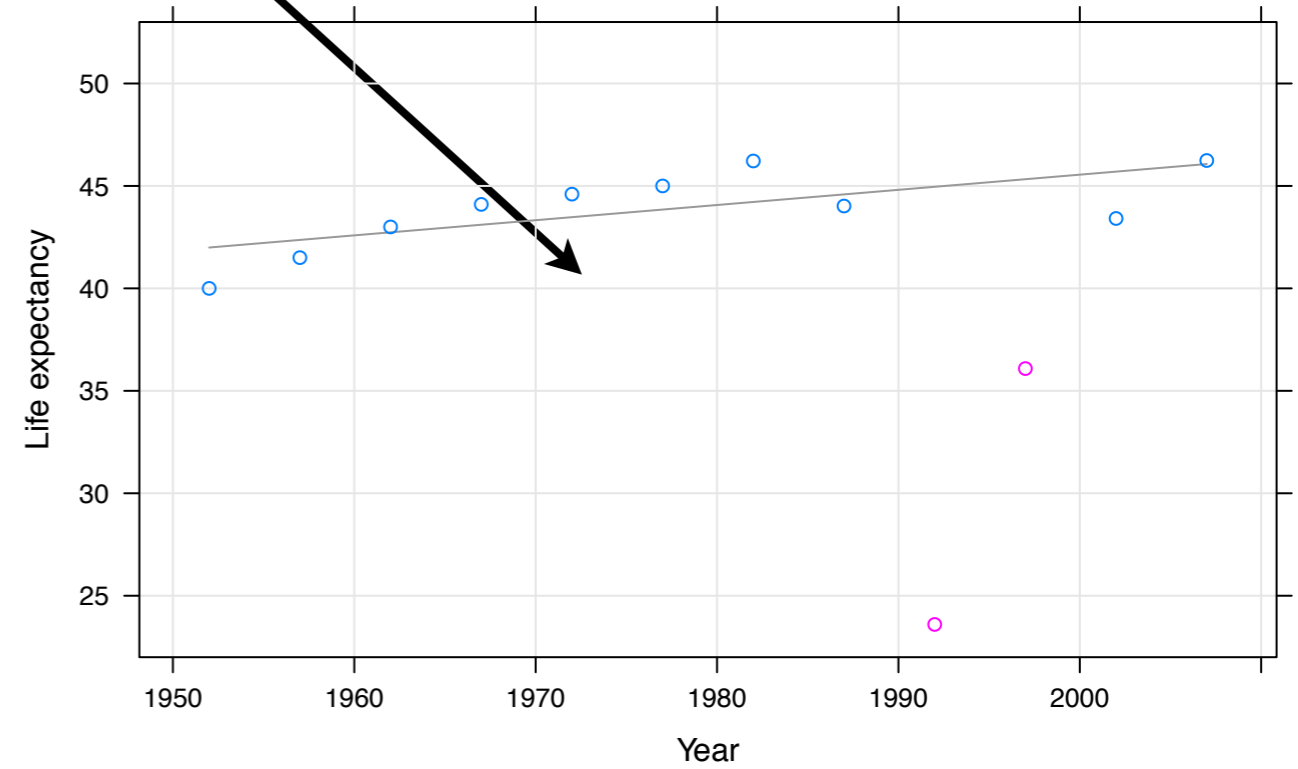


```
> summary(lsFit)
```

```
Call:  
lm(formula = lifeExp ~ I(year - yearMin), data = hDat)
```

```
Residuals:  
    Min      1Q  Median      3Q     Max  
-17.310  -1.445   2.410   3.073   6.021
```

```
Coefficients:  
                Estimate Std. Error t value Pr(>|t|)  
(Intercept)    42.74195    3.56128  12.002 2.92e-07 ***  
I(year - yearMin) -0.04583    0.10969  -0.418  0.685
```



```
> summary(lsFitAlt)
```

```
Call:  
lm(formula = lifeExp ~ I(year - yearMin), data = hDat,  
    subset = flag == "data OK")
```

```
Residuals:  
    Min      1Q  Median      3Q     Max  
-2.2855 -0.7903   0.2190   1.0917   2.0012
```

```
Coefficients:  
                Estimate Std. Error t value Pr(>|t|)  
(Intercept)    41.99422    0.81870  51.294 2.31e-11 ***  
I(year - yearMin) 0.07408    0.02725   2.718  0.0263 *
```

Robust/resistant regression

- Scenario: you are concerned about the undue influence of a handful of observations, but find it unappealing to explicitly identify ‘good’ and ‘bad’ data.
- Consider our ‘default’ estimator of the parameter β in a linear model as the minimizer of the mean of the squared residuals.
- Median and trimmed mean are helpful when dealing with ‘less than perfect’ data -- use them here to construct more robust estimators of β .

Robust/resistant regression

- The construction of various robust estimators is beyond the scope of this course, but we can still use one or two robust estimators responsibly.
- Also, the sampling distribution of robust estimators will be, in general, less well-characterized than in the plain vanilla least squares or normal errors case. This will give us another opportunity to use the bootstrap.

$$y_i = \beta_0 + \beta_1 year_i + \varepsilon_i$$

estimator	notes & R code snippets
<p>LS = least (mean) squares</p>	<pre>lsFit <- lm(lifeExp ~ I(year - yearMin), hDat) lsFitAlt <- lm(lifeExp ~ I(year - yearMin), hDat, subset = flag == 'data OK')</pre>
<p>LTS = least trimmed mean squares</p>	<pre>ltsFit <- ltsReg(lifeExp ~ I(year - yearMin), hDat) ltsFitAlt <- ltsReg(lifeExp ~ I(year - yearMin), hDat, subset = flag == 'data OK')</pre> <p>ltsReg() is in the robustbase package.</p>
<p>MM = MM estimation</p>	<pre>mmFit <- lmrob(lifeExp ~ I(year - yearMin), hDat) mmFitAlt <- lmrob(lifeExp ~ I(year - yearMin), hDat, subset = flag == 'data OK')</pre> <p>lmrob() is in the robustbase package.</p> <p>This is probably what you should actually use in practice! LTS regression is included only because it can be explained / motivated rather easily, unlike MM estimation.</p>

$$y_i = \beta_0 + \beta_1 year_i + \varepsilon_i$$

Various estimates of β_1

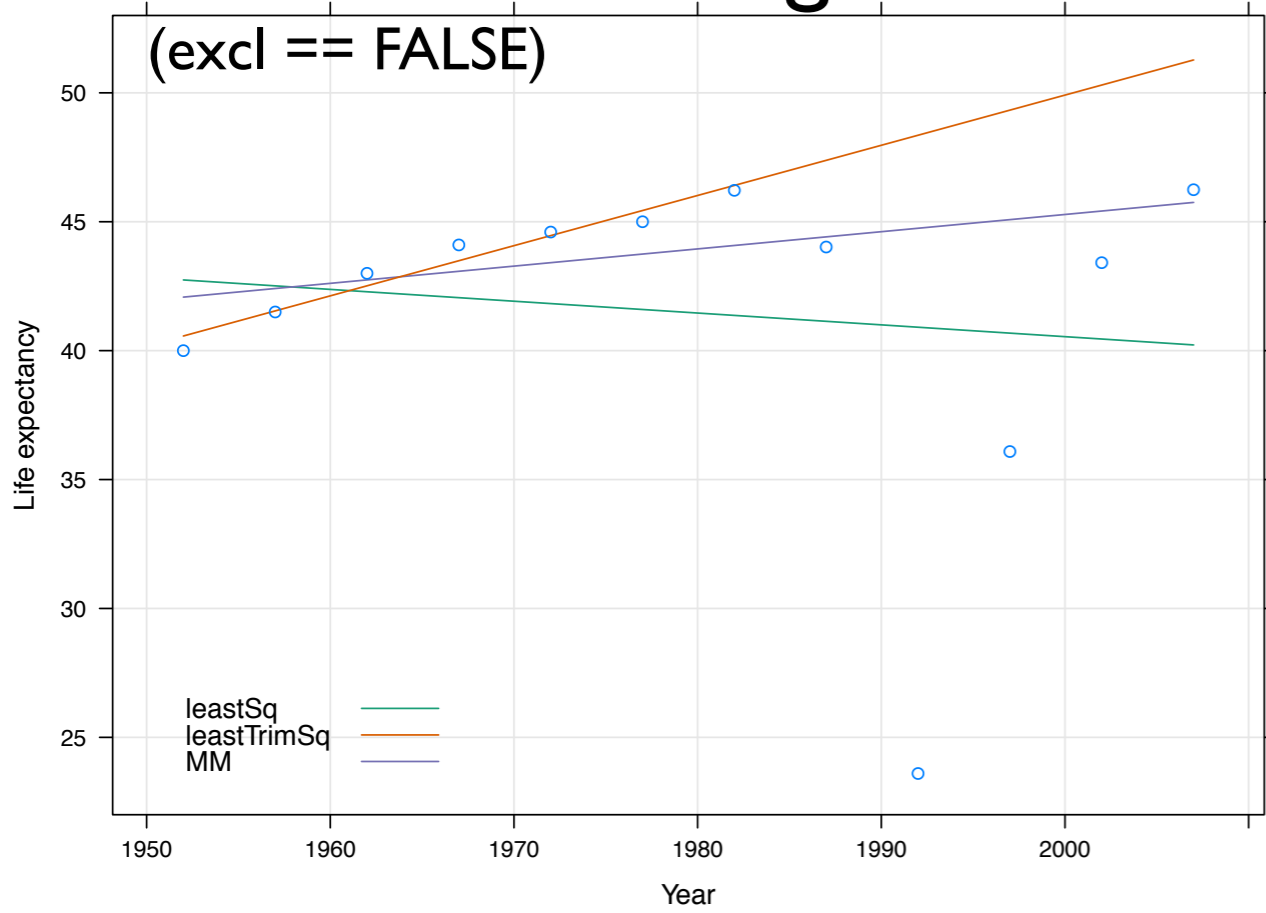
	meth	excl	est	se
1	leastSq	FALSE	-0.04583147	0.10968601
2	leastSq	TRUE	0.07408486	0.02725230
3	leastTrimSq	FALSE	0.19467143	0.01835124
4	leastTrimSq	TRUE	0.19467143	0.01835124
5	MM	FALSE	0.06679455	0.04305619
6	MM	TRUE	0.07647548	0.03494911

that table of numbers makes it really
easy to see what's going on, right?

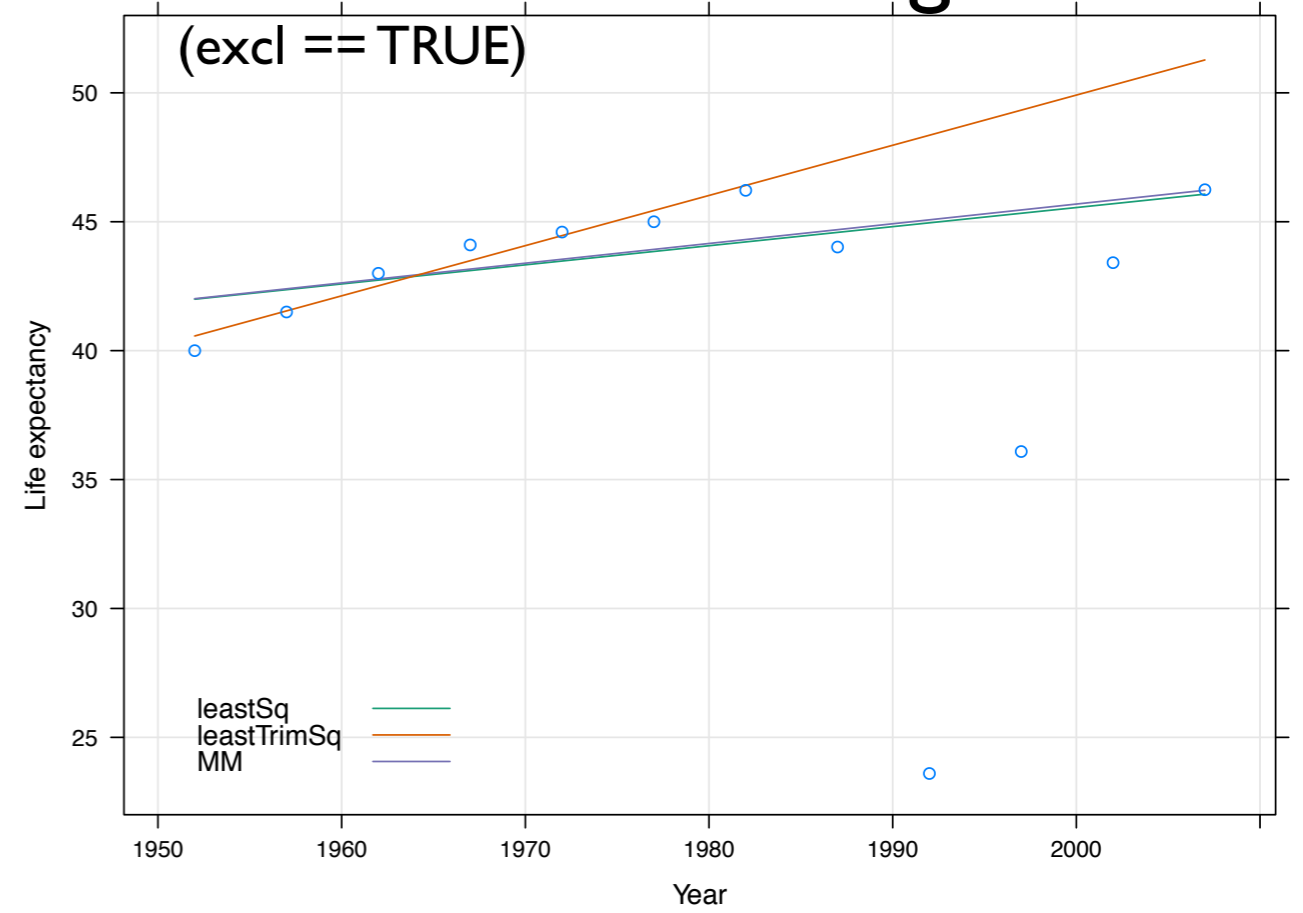
just kidding!

of course, I've made some pictures

Fitted lines using all data



Fitted lines omitting 2 outliers



meth	excl	slope est	se
leastSq	FALSE	-0.046	0.1097
leastSq	TRUE	0.074	0.0273
leastTrimSq	FALSE	0.195	0.0184
leastTrimSq	TRUE	0.195	0.0184
MM	FALSE	0.067	0.0431
MM	TRUE	0.076	0.0349

- Results for this data are encouraging
 - robust estimates are close to LS estimate when outliers are removed.
 - robust estimates are essentially same with and without outliers.
- But how are we going to do inference? It turns out that there is now a theoretical basis for providing estimated standard errors for the LTS and MM estimators. This was not always the case, mind you. But let's use the bootstrap anyway just to double-check

Bootstrapping in a regression context

$$y_i = x_i\beta + \varepsilon_i$$

- The probability model for Y has parameter (β, F_ε) , where F_ε is the distribution of the error terms ε .
- We have many ways to estimate β , but the bootstrap will require us to estimate F_ε as well.
- Option #1: estimate F_ε with the empirical distribution of the residuals.
- So, what does the bootstrap procedure look like?

Generate a size n bootstrap sample of residuals by resampling with replacement from the observed residuals.

$$\hat{\varepsilon}_i = y_i - x_i \hat{\beta}$$

$$\hat{F} : \text{probability } 1/n \text{ on } \hat{\varepsilon}_i$$

$$\hat{F} \rightarrow (\varepsilon_1^*, \varepsilon_2^*, \dots, \varepsilon_n^*)$$

Generate bootstrap data by adding the linear predictor and the bootstrap residuals.

$$y_i^* = x_i \hat{\beta} + \varepsilon_i^*$$

Generate bootstrap statistic or estimator by the same procedure used with observed data.

$$\hat{\beta}^* = \min^{-1} f[(y_i^* - x_i \beta)^2]$$

Generate bootstrap statistic or estimator by the same procedure used with observed data.

$$\hat{\beta}^* = \min^{-1} f[(y_i^* - x_i\beta)^2]$$

The function f above is determined by the choice of criterion, i.e. least squares vs. least median squares, vs. least trimmed mean, etc.

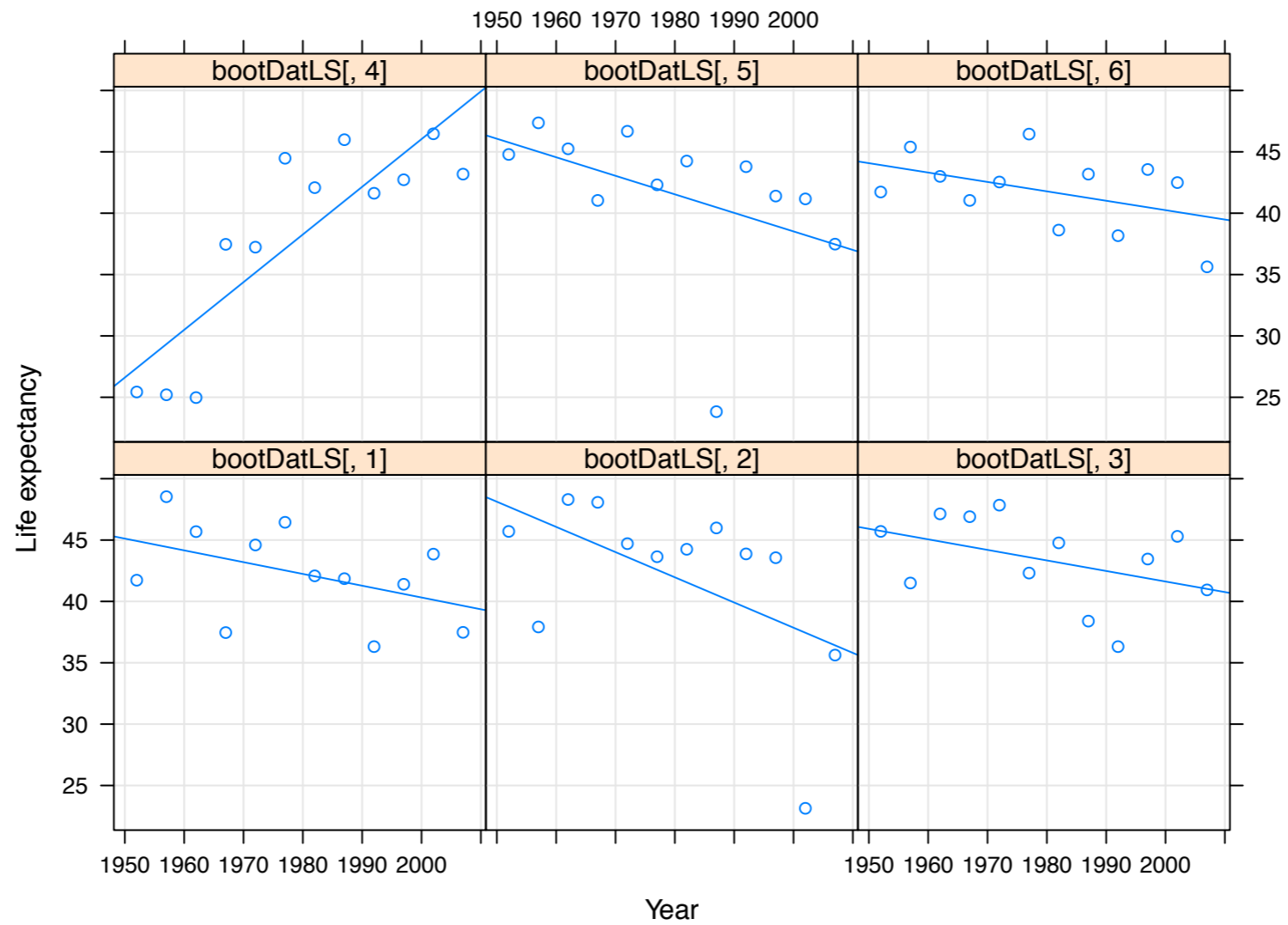
The sampling distribution of the bootstrap statistics (or estimates) can be used, for example, to estimate the standard error of the observed statistic or estimate.

Hypothesis testing is also possible -- but recall you must explicitly enforce the null hypothesis!

In fact, the bootstrap can take very many forms. In particular, different assumptions made about F , the distribution of the errors, will produce different bootstrap approaches.

I have briefly introduced 'bootstrapping residuals', but it is also possible to 'bootstrap pairs' (option #2).

The first six bootstrap datasets, with least squares fits.



```
B <- 1000
set.seed(17)
bootIndices <- sample(1:n, n * B, replace = TRUE)
```

**resample the
observations**

```
> bootIndices[1:(3 * n)]
 [1]  2 12  6 10  5  7  3  3 10  3  6  1 11 10 12 12  8  4  8  7 11  8  9 10 11
[26]  2  7  7 12  3  6  1 10  5  7  3
```

```
> matrix(resid(lsFit)[bootIndices[1:(3 * n)]], nrow = n)
```

```
      [,1]      [,2]      [,3]
[1,] -1.012791  2.962625  2.962625
[2,]  6.020782 -4.592533 -1.012791
[3,]  3.403838  6.020782  4.850995
[4,] -4.592533  6.020782  4.850995
[5,]  2.774681  2.882153  6.020782
[6,]  4.850995  2.045523  0.716366
[7,]  0.716366  2.882153  3.403838
[8,]  0.716366  4.850995 -2.741949
[9,] -4.592533  2.962625 -4.592533
[10,]  0.716366  2.882153  2.774681
[11,]  3.403838 -17.309690  4.850995
[12,] -2.741949 -4.592533  0.716366
```

**grab the associated
residuals and store as
matrix, 1 column per
bootstrap dataset**

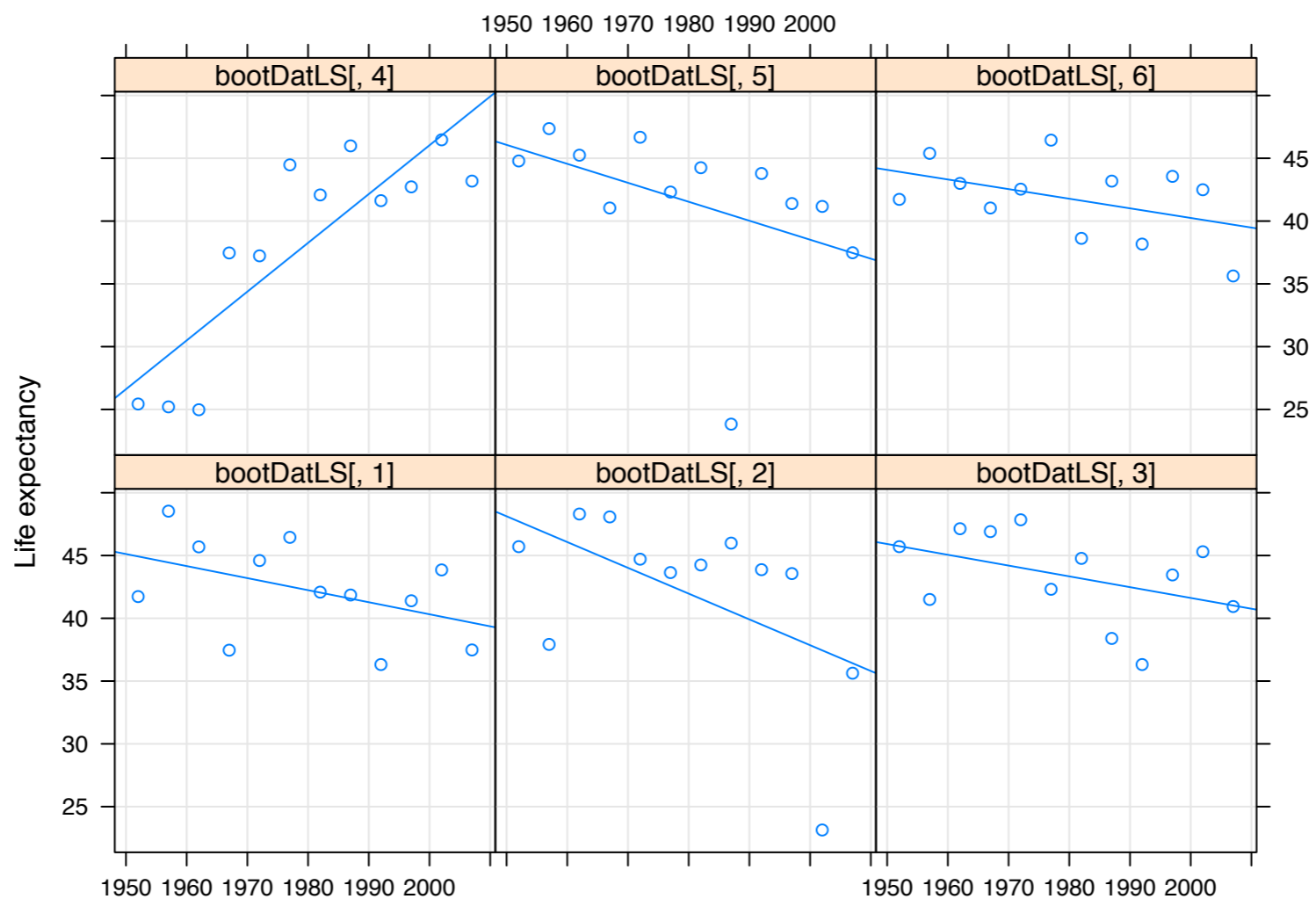
```
sweep(matrix(resid(lsFit)[bootIndices[1:(3 * n)]], nrow = n), 1,
      fitted(lsFit), "+")
```

	[,1]	[,2]	[,3]		[,1]		[,1]	[,2]	[,3]
[1,]	-1.012791	2.962625	2.962625	1285	42.74195	[1,]	41.72916	45.70457	45.70457
[2,]	6.020782	-4.592533	-1.012791	1286	42.51279	[2,]	48.53357	37.92026	41.50000
[3,]	3.403838	6.020782	4.850995	1287	42.28363	[3,]	45.68747	48.30442	47.13463
[4,]	-4.592533	6.020782	4.850995	1288	42.05448	[4,]	37.46194	48.07526	46.90547
[5,]	2.774681	2.882153	6.020782	1289	41.82532	[5,]	44.60000	44.70747	47.84610
[6,]	4.850995	2.045523	0.716366	1290	41.59616	[6,]	46.44716	43.64169	42.31253
[7,]	0.716366	2.882153	3.403838	1291	41.36700	[7,]	42.08337	44.24916	44.77084
[8,]	0.716366	4.850995	-2.741949	1292	41.13785	[8,]	41.85421	45.98884	38.39590
[9,]	-4.592533	2.962625	-4.592533	1293	40.90869	[9,]	36.31616	43.87131	36.31616
[10,]	0.716366	2.882153	2.774681	1294	40.67953	[10,]	41.39590	43.56169	43.45421
[11,]	3.403838	-17.309690	4.850995	1295	40.45038	[11,]	43.85421	23.14069	45.30137
[12,]	-2.741949	-4.592533	0.716366	1296	40.22122	[12,]	37.47927	35.62869	40.93758

add the residuals to
the fit to obtain
bootstrap data

```
sweep(matrix(resid(lsFit)[bootIndices[1:(3 * n)]], nrow = n), 1,
      fitted(lsFit), "+")
```

	[,1]	[,2]	[,3]		[,1]		[,1]	[,2]	[,3]
[1,]	-1.012791	2.962625	2.962625	1285	42.74195	[1,]	41.72916	45.70457	45.70457
[2,]	6.020782	-4.592533	-1.012791	1286	42.51279	[2,]	48.53357	37.92026	41.50000
[3,]	3.403838	6.020782	4.850995	1287	42.28363	[3,]	45.68747	48.30442	47.13463
[4,]	-4.592533	6.020782	4.850995	1288	42.05448	[4,]	37.46194	48.07526	46.90547
[5,]	2.774681	2.882153	6.020782	1289	41.82532	[5,]	44.60000	44.70747	47.84610
[6,]	4.850995	2.045523	0.716366	1290	41.59616	[6,]	46.44716	43.64169	42.31253
[7,]	0.716366	2.882153	3.403838	1291	41.36700	[7,]	42.08337	44.24916	44.77084
[8,]	0.716366	4.850995	-2.741949	1292	41.13785	[8,]	41.85421	45.98884	38.39590
[9,]	-4.592533	2.962625	-4.592533	1293	40.90869	[9,]	36.31616	43.87131	36.31616
[10,]	0.716366	2.882153	2.774681	1294	40.67953	[10,]	41.39590	43.56169	43.45421
[11,]	3.403838	-17.309690	4.850995	1295	40.45038	[11,]	43.85421	23.14069	45.30137
[12,]	-2.741949	-4.592533	0.716366	1296	40.22122	[12,]	37.47927	35.62869	40.93758



Perform all “sampling with replacement” at once.

```
> B <- 1000
> bootIndices <- sample(1:n, n * B, replace = TRUE)

> bootDatLS <-
+ sweep(matrix(resid(lsFit)[bootIndices], nrow = n), 1,
+ fitted(lsFit), "+")
```

Generate bootstrap residuals: index resid by random indices.

Give bootstrap residuals correct shape = n by B matrix.

Generate bootstrap data with the `sweep()` call: Adds the vector of fitted values to each column of `bootDat`, i.e. to each vector-valued bootstrap residual. Read up on `sweep()` -- very handy.

$$\hat{\varepsilon}_i = y_i - x_i \hat{\beta}$$

Concept: \hat{F} : probability $1/n$ on $\hat{\varepsilon}_i$

$$\hat{F} \rightarrow (\varepsilon_1^*, \varepsilon_2^*, \dots, \varepsilon_n^*)$$

Implementation:

```
bootIndices <- sample(1:n, n * B, replace = TRUE)
resid(lsFit)[bootIndices]
```

Concept: $y_i^* = x_i \hat{\beta} + \varepsilon_i^*$

Implementation:

```
bootDatLS <-
  sweep(matrix(resid(lsFit)[bootIndices], nrow = n), 1,
        fitted(lsFit), "+")
```

Use `apply()` to visit each bootstrap dataset (i.e. column) and retain estimated slope.



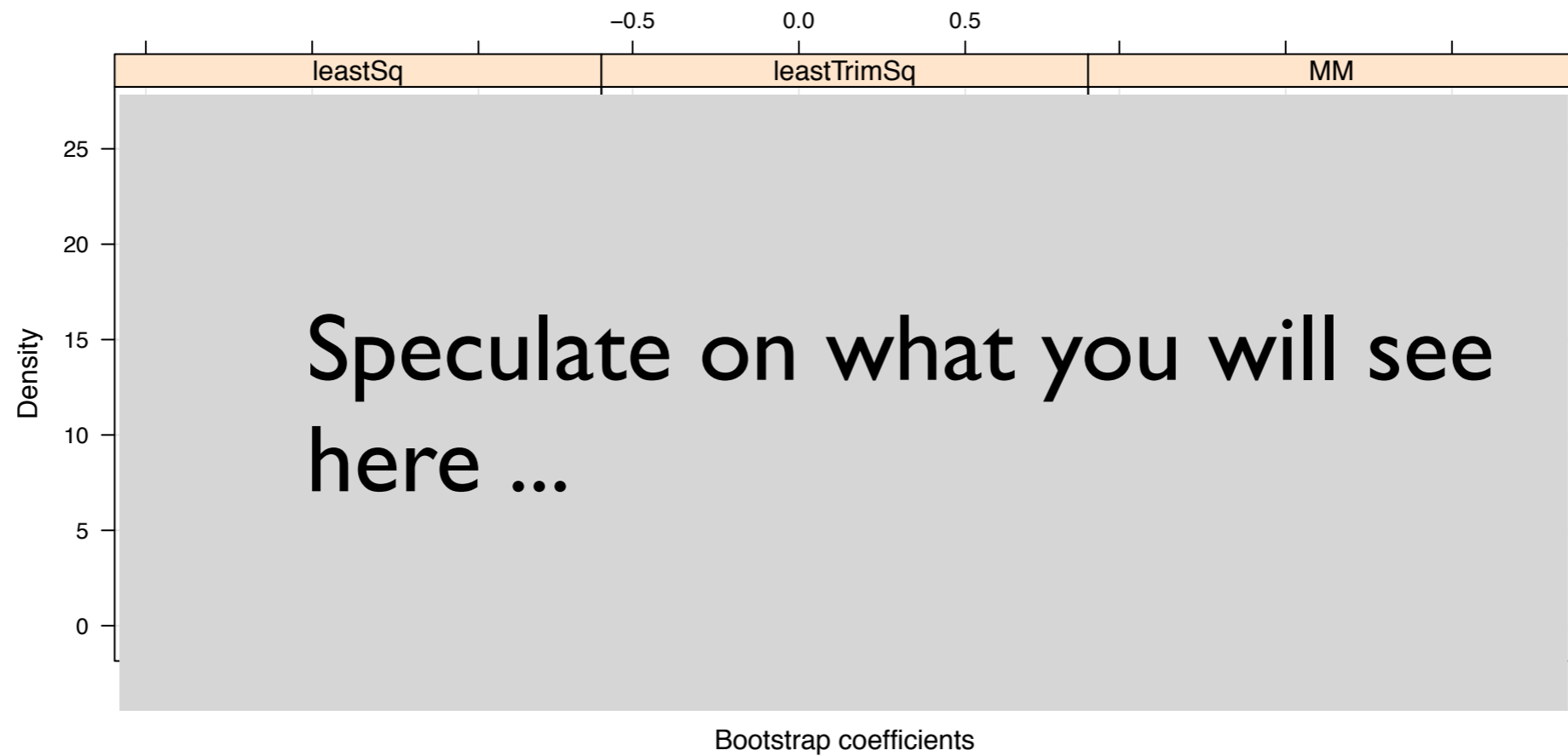
```
bootCoefLS <-  
  apply(bootDatLS, 2, function(lifeExp) {  
    coef(lm(lifeExp ~ I(jYear - yearMin))["I(jYear - yearMin)"]  
  })  
  
> str(bootCoefLS)  
num [1:1000] -0.096 -0.2053 -0.0859 0.3885 -0.151 ...
```

Concept: $\hat{\beta}_{LS}^* = \min^{-1} \text{mean}_{\tau} [(\varepsilon_i^*)^2]$

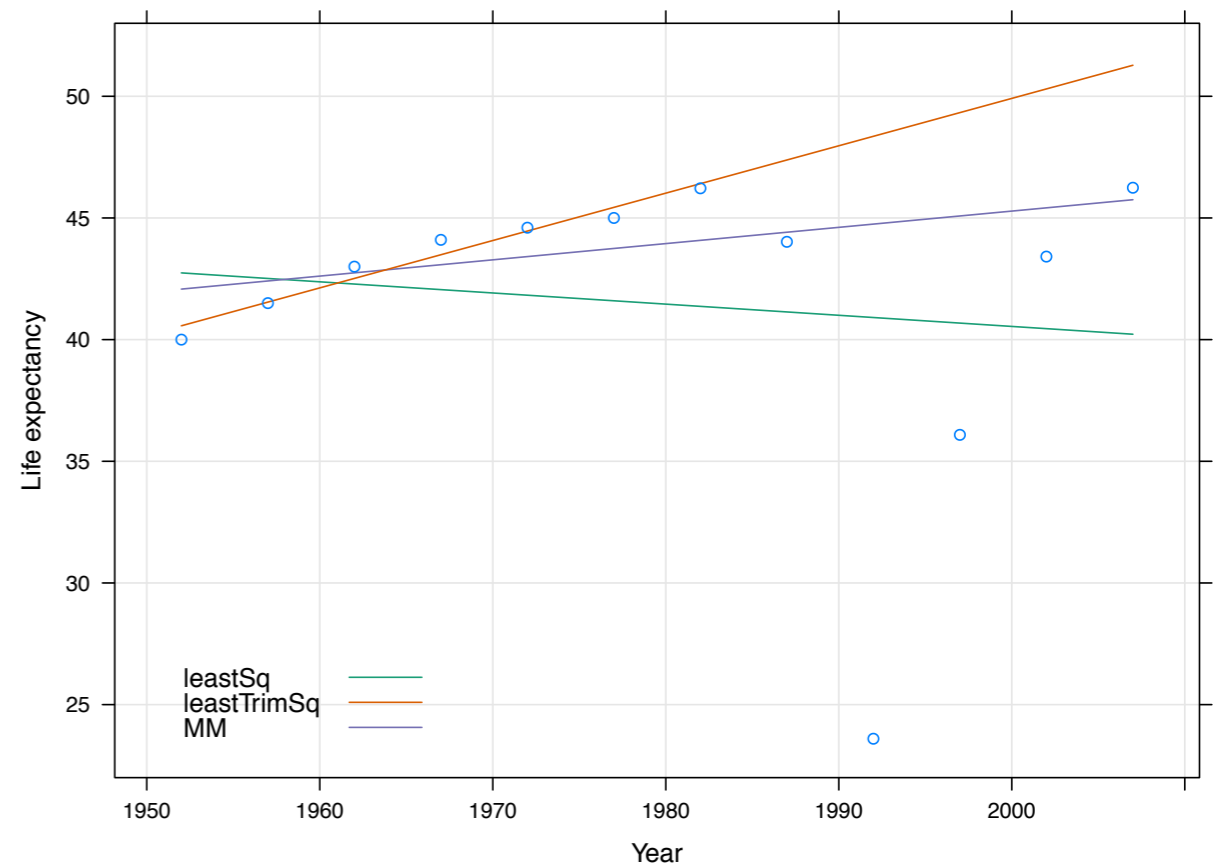
Implementation:

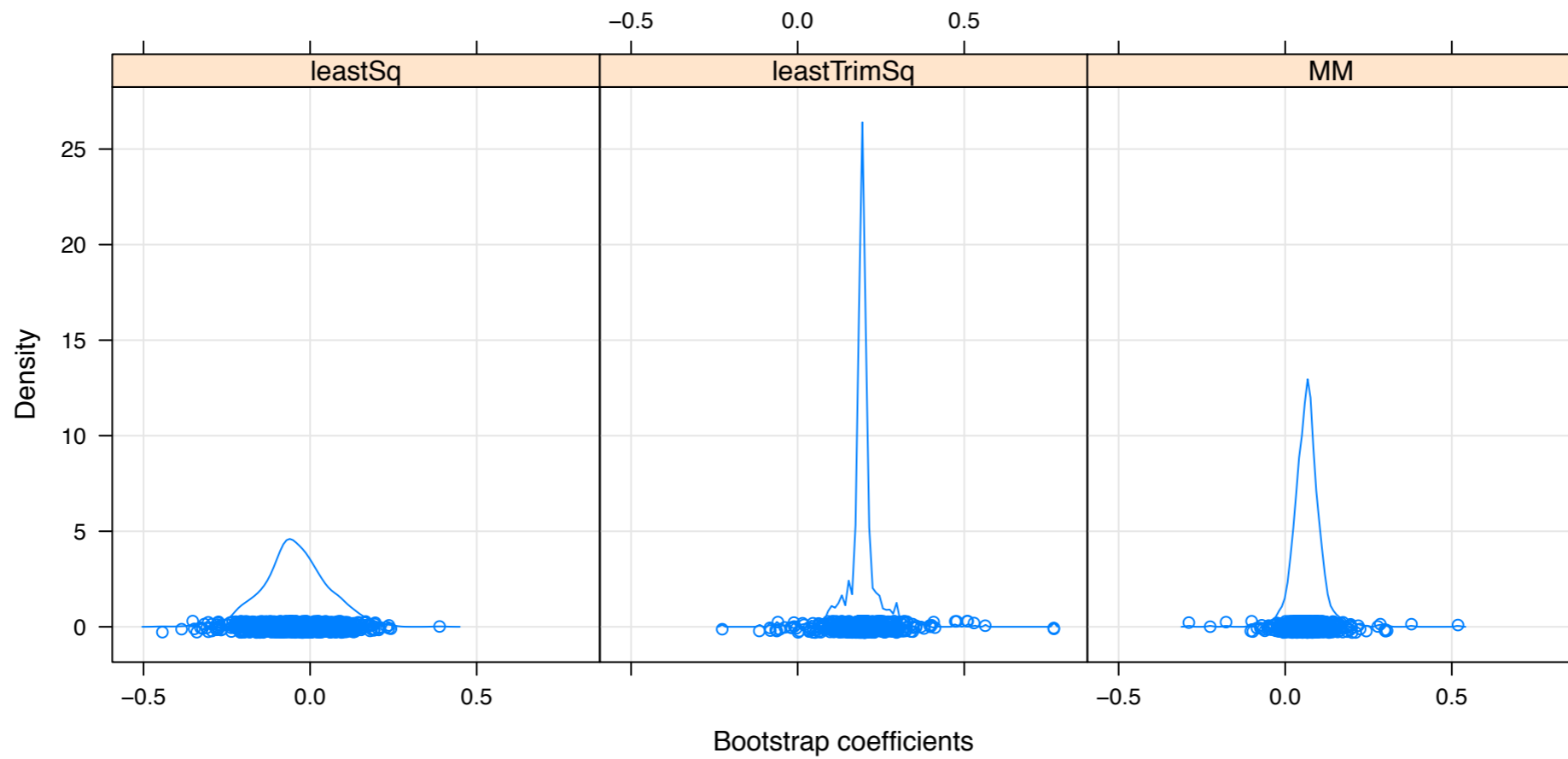
```
bootCoefLS <-  
  apply(bootDatLS, 2, function(lifeExp) {  
    coef(lm(lifeExp ~ I(jYear - yearMin)))[ "I(jYear - yearMin)" ]  
  })
```

Now, let's look at the bootstrap distribution of the least squares, least trimmed squares, and MM slope estimates.

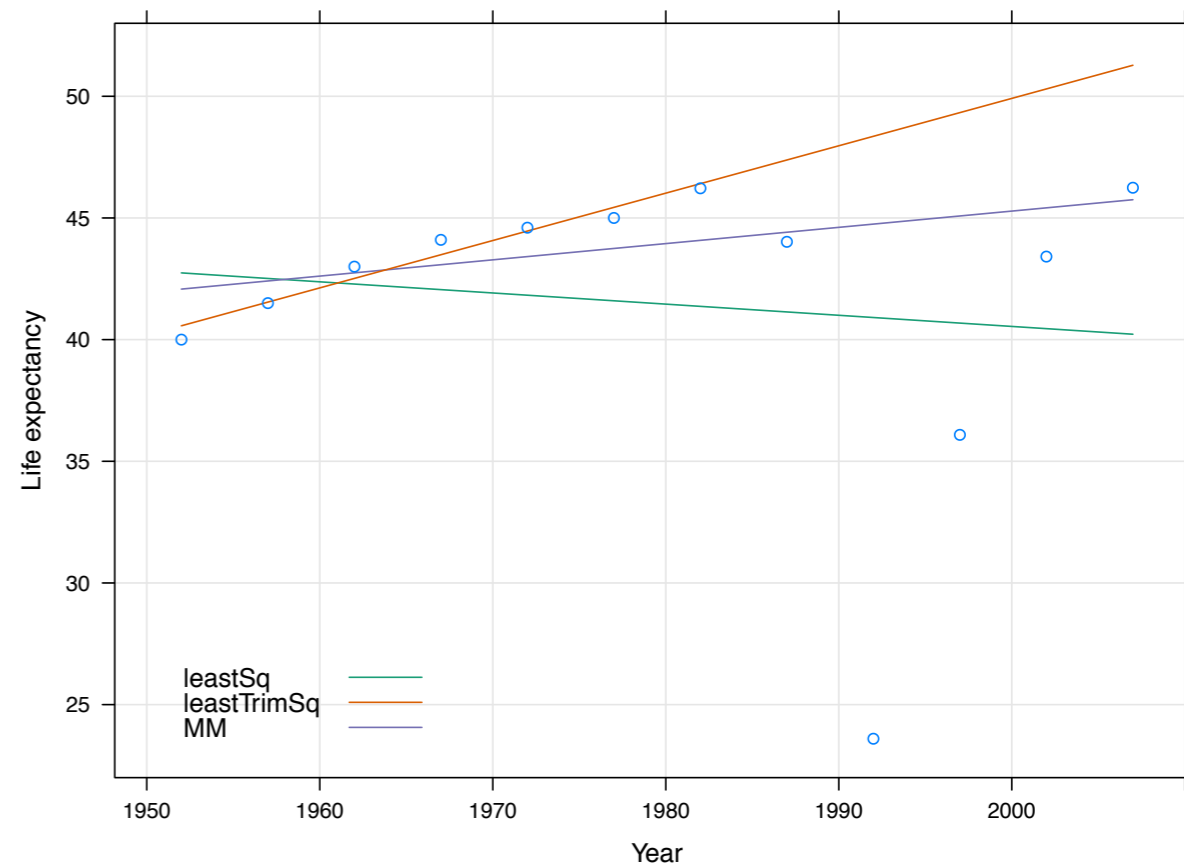


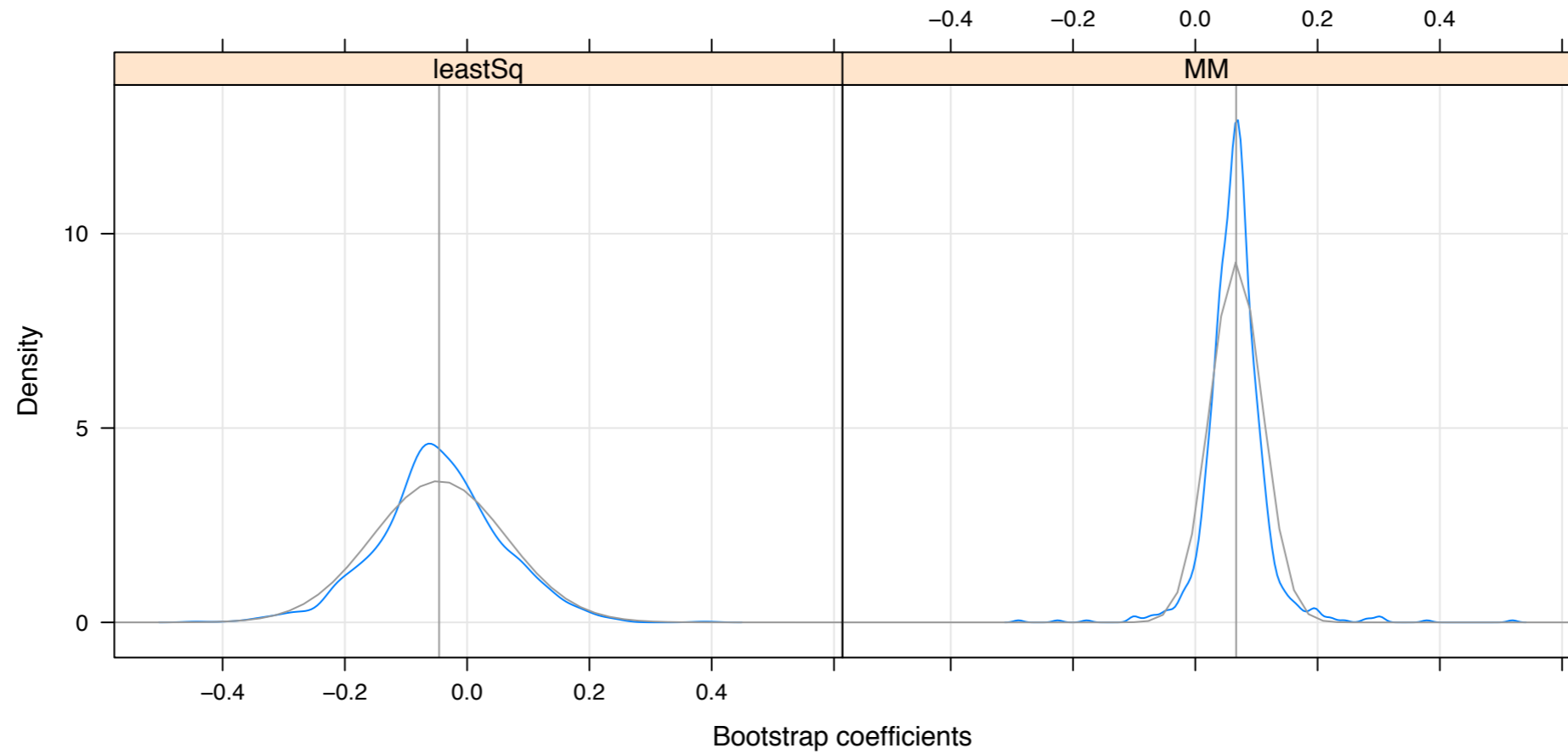
meth	est	se
leastSq	-0.046	0.1097
leastTrimSq	0.195	0.0184
MM	0.067	0.0431



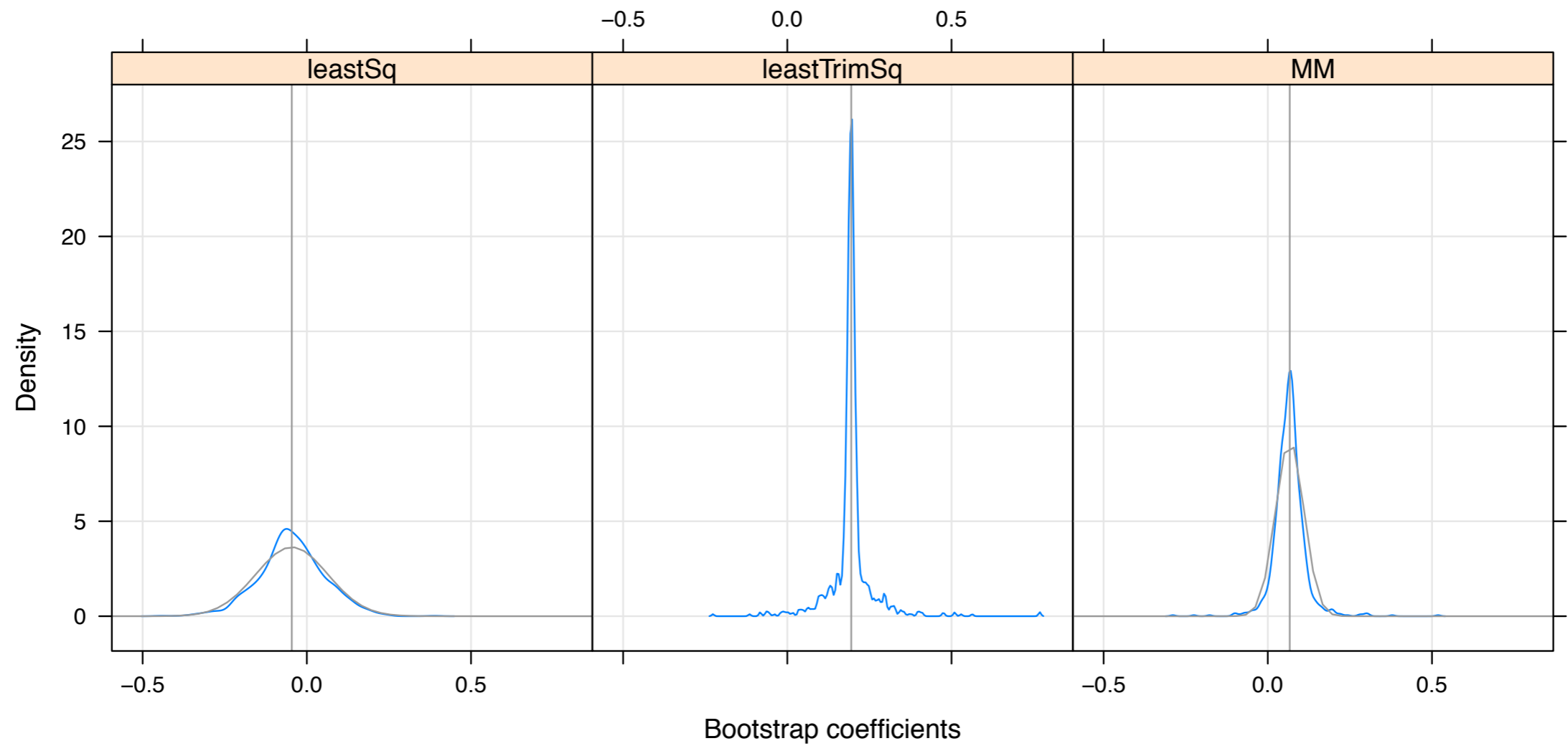


meth	est	se
leastSq	-0.046	0.1097
leastTrimSq	0.195	0.0184
MM	0.067	0.0431





Both the LS and MM estimators are asymptotically normal. This asymptotic distribution is superposed in grey.



meth	est	avgBoot	se	seBoot
leastSq	-0.046	-0.045	0.1097	0.101
leastTrimSq	0.195	0.194	0.0184	0.0659
MM	0.067	0.066	0.0431	0.0494

What have we learned from this example?

- In this case, the robust method (MM) seems to produce more “correct” result, i.e. that life expectancy in Rwanda is flat or gradually increasing, modulo the early/mid 1990s.
 - I believe we are “at real risk of deceiving ourselves” if we use LS estimation.
- Bootstrap gives us a way to conduct inference with non-standard estimators and to check the relevance of asymptotic results when available.
- Good to try different methods -- “all models are wrong”.
- Bootstrapping can (and should) be carried out with no explicit looping.

JB briefly discussed a “mechanics” issue most of you will encounter in your projects: **data reshaping**.

In case you were wondering: no, you are not crazy or doing something wrong. The need to reshape data actually arises quite often.

Usually the need to fit a model or make a figure is what triggers this.

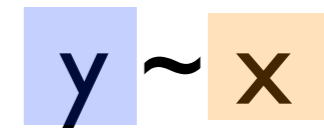
Slides are crude but here just to record the conversation.

Demo of why/how to reshape data (crime data riddell & ushey used for an assignment in 2009)

```
> str(cDat)
'data.frame':   17 obs. of  6 variables:
 $ Year      : int  1990 1991 1992 1993 1994 1995 1996 ...
 $ Vcrime    : num  4.15 4.17 4.28 4.24 3.97 ...
 $ Pcrime    : num  24.6 26.7 25.6 23.9 22.8 ...
 $ UnempRate : num  0.0544 0.072 0.0709 0.0695 0.066 0.0595 0.0582 ...
 $ PercentLowInc: num  0.125 0.127 0.142 0.142 0.15 0.149 0.167 ...
 $ PercentGrad : num  0.685 0.697 0.703 0.715 0.718 ...
```

```
> cDat
```

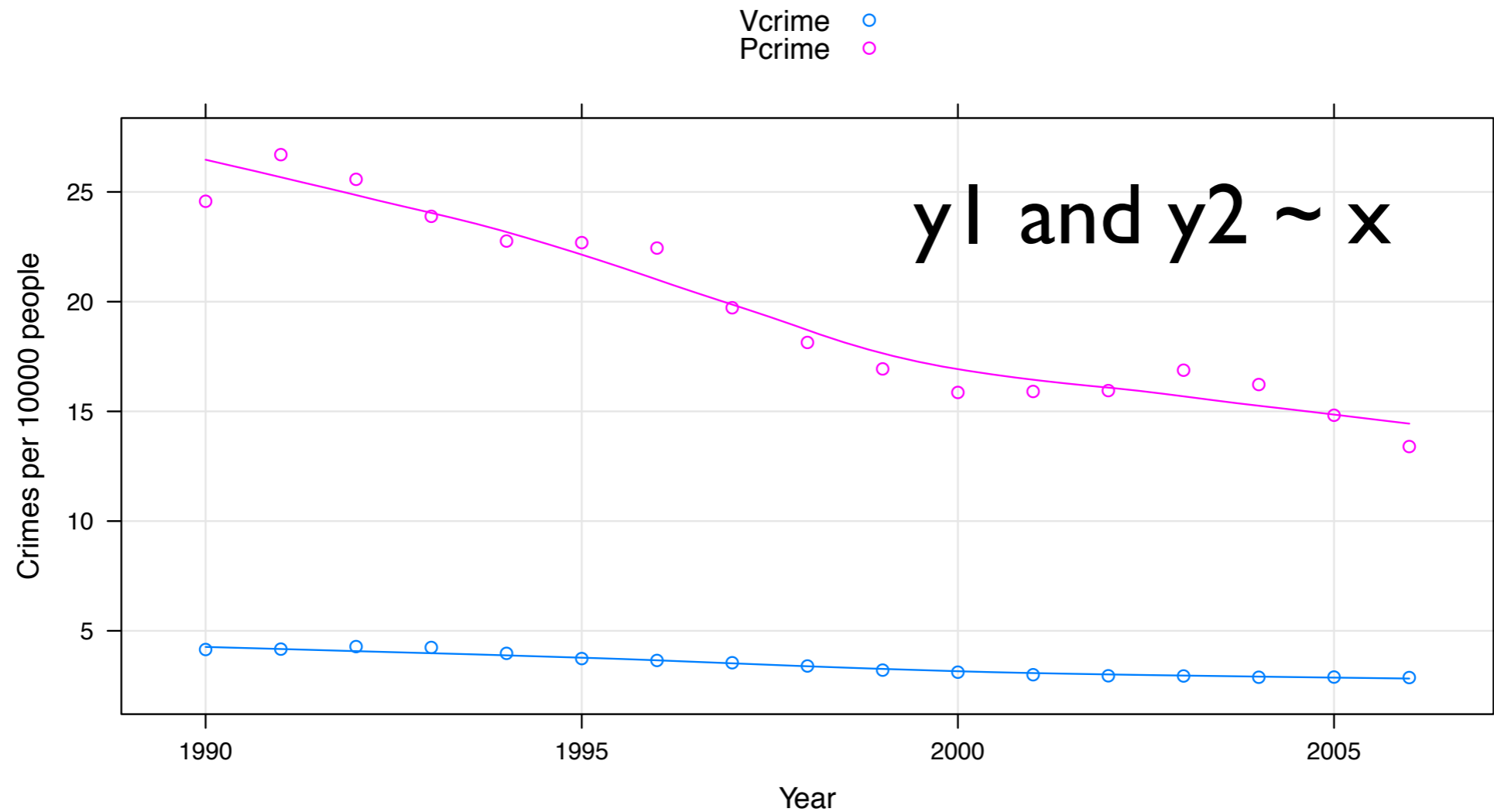
	Year	Vcrime	Pcrime	UnempRate	PercentLowInc	PercentGrad
1	1990	4.148524	24.57420	0.0544	0.125	0.6847
2	1991	4.167157	26.69928	0.0720	0.127	0.6974
3	1992	4.282401	25.57468	0.0709	0.142	0.7032
4	1993	4.241863	23.88863	0.0695	0.142	0.7152
5	1994	3.974130	22.76257	0.0660	0.150	0.7180
6	1995	3.735833	22.69167	0.0595	0.149	0.7212
7	1996	3.650269	22.44285	0.0582	0.167	0.7314
8	1997	3.546817	19.72201	0.0538	0.165	0.7305
9	1998	3.397468	18.14151	0.0631	0.146	0.7363
10	1999	3.209997	16.93544	0.0566	0.164	0.7378
11	2000	3.116337	15.86382	0.0480	0.151	0.7505
12	2001	3.000394	15.90761	0.0540	0.141	0.7585
13	2002	2.951312	15.94828	0.0580	0.160	0.7652
14	2003	2.942124	16.87751	0.0535	0.154	0.7781
15	2004	2.885056	16.22376	0.0483	0.141	0.7869
16	2005	2.894142	14.82477	0.0380	0.130	0.7923
17	2006	2.870854	13.39746	0.0308	0.130	0.7935



Vcrime and Pcrime report crime rates for violent and property crime respectively.

What if you want to plot both against year in the same panel?

First the quick-and-dirty solution is to be lazy and avoid reshaping the data. Rely on lattice's "extended formula interface".



```
## using 'extended formula interface' in lattice
xyplot(Vcrime + Pcrime ~ Year, cDat,
       type = c('p', 'smooth', 'g'), auto.key = TRUE,
       ylab = paste("Crimes per", rateCons, "people"))
## default for y1 + y2 ~ x is to superpose y1 and y2 on same
## scatterplot
## looks like (and is handled internally) like the result of using the
## 'groups' argument
```

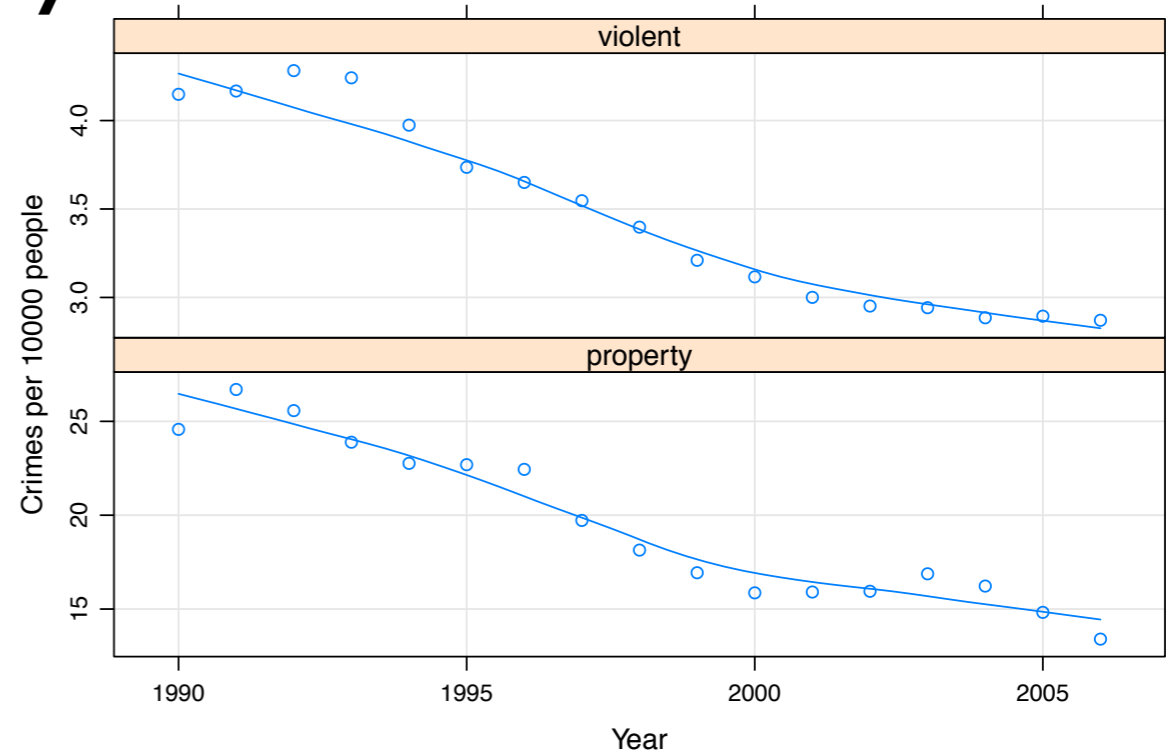
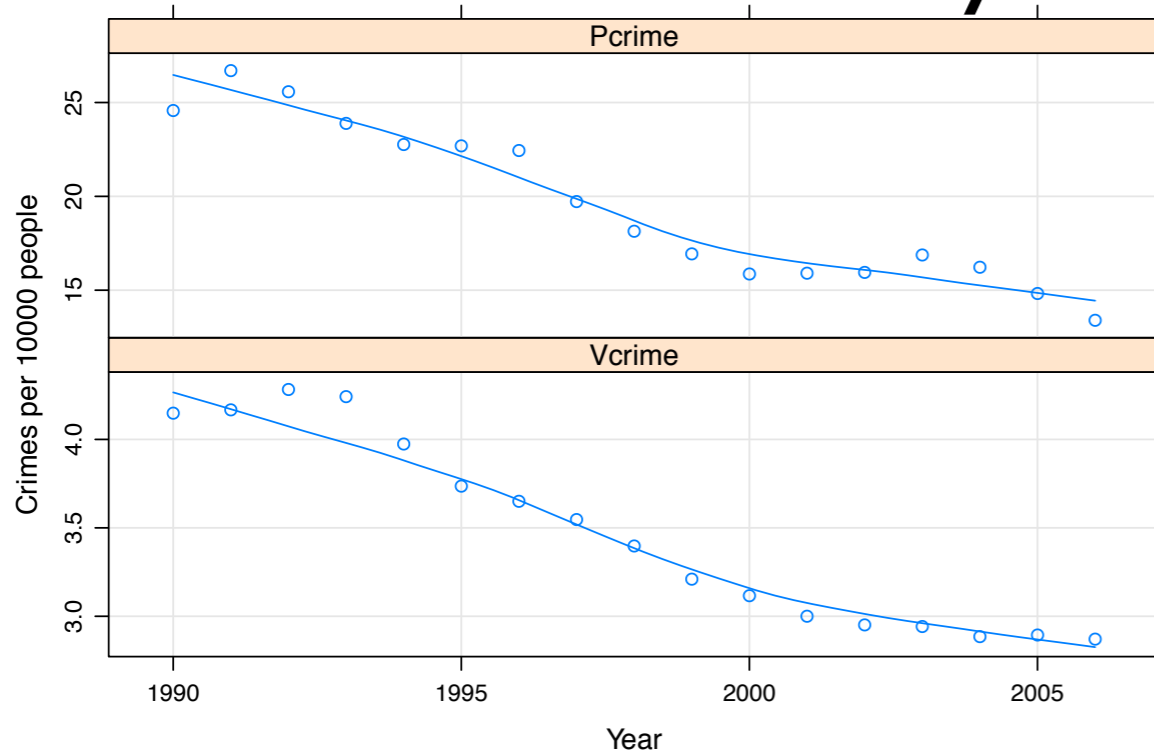
**‘extended formula interface’ in lattice;
superposes by default**

Now ...
what if you want to plot each
crime rate in its own panel?

First the quick-and-dirty
solution is to be lazy and avoid
reshaping the data. Rely on
lattice's "extended formula
interface".

To get "conditioning" in panels
instead of superposition a la
groups, use the argument
"outer = TRUE".

y_1 and $y_2 \sim x$



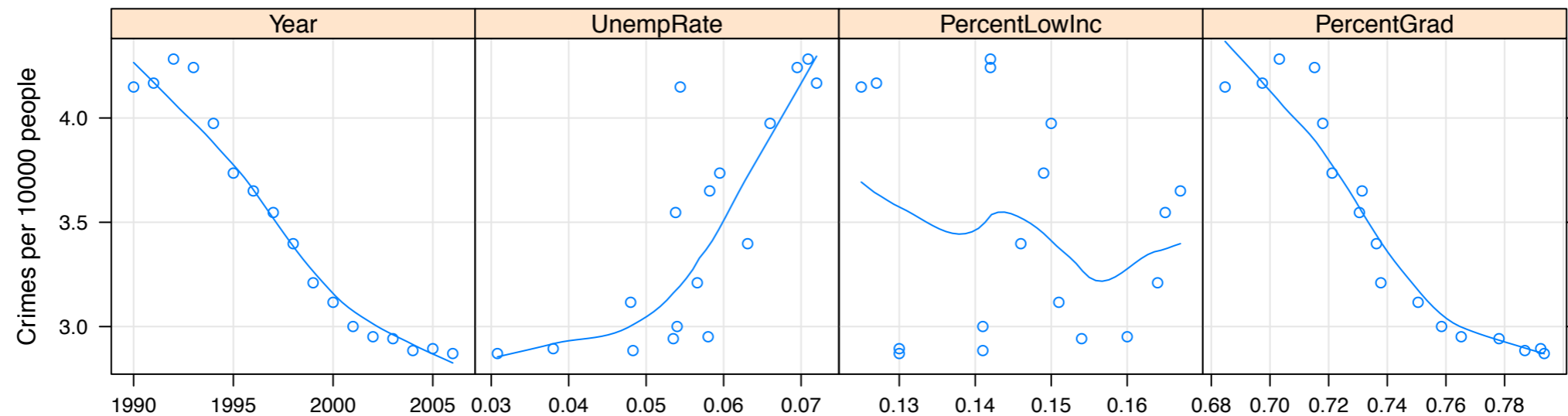
```
## using 'extended formula interface' in lattice
## AND asking for separate panels
## also requesting different y axis limits
xyplot(Vcrime + Pcrime ~ Year, cDat,
       type = c('p', 'smooth', 'g'),
       outer = TRUE, layout = c(1, 2),
       ylab = paste("Crimes per", rateCons, "people"),
       scales = list(y = list(relation = 'free')))
```

‘extended formula interface’
in lattice; can simulate a
conditioning plot

‘proper’ conditioning plot,
after reshaping data

```
xyplot(crimeRate ~ Year | crimeType, dDat,
       type = c('p', 'smooth', 'g'),
       layout = c(1, 2),
       ylab = paste("Crimes per", rateCons, "people"),
       scales = list(y = list(relation = 'free')))
```

$y_1 \sim x_1$ and x_2 and x_3 and x_4



```
## look at relationship of crime to socioeconomic predictors
## using 'extended formula interface' in lattice again
## AND asking for separate panels
xyplot(Vcrime ~ Year + UnempRate + PercentLowInc + PercentGrad, cDat,
       type = c('p', 'smooth', 'g'),
       outer = TRUE, layout = c(4, 1), xlab = "",
       ylab = paste("Crimes per", rateCons, "people"),
       scales = list(x = list(relation = 'free')))
```

‘extended formula interface’ in lattice; it even works for
x too!

Bottom line:

Extended formula interface works and can be handy

Actual output with proper conditioning plots, based on reshaped data, can be a little bit better (options for 'relation' of axis 'scales' still not perfect either way)

Code for proper conditioning plots, based on reshaped data, is more reusable -- don't have to hard-wire y_1 , y_2 , x_1 , x_2 , etc. and can query the number of factor levels to fix the layout

Lattice likes 'tall and skinny' datasets better than 'short and fat' ones. Sad fact of life.

two common scenarios when reshaping is necessary:

“raw” data has 1 row per experimental unit and repeated observations of some response, e.g. over time, is found in several columns

some apply-type function has returned something matrix-like and you really wanted the rows or columns stacked on top of each other

The Common Need: you've got something that's short and fat but you wish it were tall and skinny.

how to actually reshape data

original

	Year	Vcrime	Pcrime	UnempR
1	1990	4.148524	24.57420	0.0
2	1991	4.167157	26.69928	0.0
3	1992	4.282401	25.57468	0.0
...				
15	2004	2.885056	16.22376	0.0483
16	2005	2.894142	14.82477	0.0380
17	2006	2.870854	13.39746	0.0308

```
dDat <-
  with(cDat,
    data.frame(Year = Year,
               Popn = Popn,
               crimeRate = c(Vcrime, Pcrime),
               crimeType = rep(c('violent', 'property'),
                              each = nrow(cDat)),
               UnempRate = UnempRate,
               PercentLowInc = PercentLowInc,
               PercentGrad = PercentGrad))
```

after 'crime' reshaping "by hand"

	Year	Popn	crimeRate	crimeType	UnempRate	PercentLowInc	PercentGrad
1	1990	3292111	4.148524	violent	0.0544	0.125	0.6847
2	1991	3373787	4.167157	violent	0.0720	0.127	0.6974
3	1992	3468802	4.282401	violent	0.0709	0.142	0.7032
...							
15	2004	4155170	2.885056	violent	0.0483	0.141	0.7869
16	2005	4196788	2.894142	violent	0.0380	0.130	0.7923
17	2006	4243580	2.870854	violent	0.0308	0.130	0.7935
18	1990	3292111	24.574202	property	0.0544	0.125	0.6847
19	1991	3373787	26.699285	property	0.0720	0.127	0.6974
20	1992	3468802	25.574680	property	0.0709	0.142	0.7032
...							
32	2004	4155170	16.223765	property	0.0483	0.141	0.7869
33	2005	4196788	14.824766	property	0.0380	0.130	0.7923
34	2006	4243580	13.397462	property	0.0308	0.130	0.7935

See also Chapter
9 of Spector
(2008)!!!!

Lowest-tech: Reshape “by hand” as on previous page. R tip: ‘recycling’ rules help you when reshaping data.frames.

Built-in functions `stack()` and `reshape()` also do this, but I find more trouble than they’re worth. JB advises do it “by hand” or move up to the next level of sophistication ...

Packages reshape and plyr by Hadley Wickham. A total solution for reshaping and data aggregation.