Functions, once again

Modified from materials by Mark Hansen, STAT 202a, and Heike Hofmann, STAT 579

Environments and function calls

Typically, once a value is returned, **the environment created to evaluate a function is "garbage collected"** -- That is, when there are no more references pointing to it, R will delete it from memory

We can use the process R follows to evaluate functions in interesting ways...

```
> power <- function(lambda)
 {
    g <- function(x) x^lambda
    return(g)
 }

# the environment associated with square() refers to where it was defined; in
# this case it was defined in a call to power()

> square <- power(2)
> cube <- power(3)

> square(1:5)
[1]  1  4  9 16 25

> cube(1:5)
[1]   1   8  27  64 125

> environment(square)
<environment: 0x16ecb34>

> environment(cube)
<environment: 0x16ec8b0>
```

Environments and function calls

Here we used **a function to create another function**; in this case, the environment of the new function (`square`) is the evaluation environment of the function we invoked (`power`)

We can see that the two functions `square` and `cube` have different environments by printing them out

The value of `lambda` used by each is stored in its environment -- Just like objects are stored in your own workspace

```
> square
function(x) x^lambda
<environment: 0x16ecb34>

> cube
function(x) x^lambda
<environment: 0x16ec8b0>

> parent.env(environment(square))
<environment: R_GlobalEnv>

> parent.env(environment(cube))
<environment: R_GlobalEnv>

> objects(environment(square))
[1] "g"      "lambda"

> objects(environment(cube))
[1] "g"      "lambda"
>

> get("lambda",env=environment(square))
[1] 2

> get("lambda",env=environment(cube))
[1] 3
```

```
> make.cdf <- function(x)
 {
   n <- length(x)
   out <- function(q) return(mean(x<=q))
   return(out)
 }

> x <- rnorm(100)

> mycdf <- make.cdf(x)

> mycdf
function(q) return(mean(x<=q))
<environment: 0x1739e0c>

> mycdf(-1)
[1] 0.16

> mycdf(0.5)
[1] 0.66

> get("n",env=environment(mycdf))
[1] 100
```

Final word on environments

As an object, we've said that **an environment is a collection of symbol-value pairs**, with an additional piece of information describing its parent

Aside from these function gymnastics, you can create environments directly and use them a bit like lists -- They are unlike lists in R in two important ways

First, you can **only access the values by name** and not by "number" as there is no sense of linear order here -- Second environments and **their contents are not copied** if they are passed as arguments to functions

This last point is crucial because it breaks all the rules we have learned so far about side effects and function evaluation -- The mechanism provides for the pass-by-reference semantics we met Python and should be used with great caution

Data-directed programming

I will use the term “data-directed programming” for the programming structures we will talk about today; in most books on R, this material is found under the heading of “Object-oriented programming”

In “classical” object-oriented programming, we see **a division between how data are represented and the operations that can sensibly be performed on them** -- This often leads to code that is easier to write and maintain

As its name suggests, object-oriented programming deals in objects, typically physical things that are represented in data by **classes** and **methods** or functions are written to manipulate them in various ways

Data-directed v. object-oriented programming

Many object-oriented languages like Python and Java are (as Gentleman describes them) **“class-centric”**, meaning they classes define objects and are “repositories for the methods that act on” them

In R, on the other hand, separates the class information from the creation of so-called generic functions and (again, quoting Gentleman) can be thought of as a **“function-centric”** system

Data-directed v. object-oriented programming

There are two built-in class and method systems in R (S3 and S4) and a number of others that are available via packages

These constructions were relatively late additions to the language and hence the two versions; S3 evolved out of a significant effort to introduce modeling functions (`lm`, `glm`, `gam`, `loess`) into the language in the early 1990s, while S4 was, well, a more reliable second attempt in the late 1990s

S3 lacks formal specification of classes, and **is really about function dispatch**, about generic functions and polymorphism (we will call this **data-directed programming**); S4 introduces **formal class definitions and a complete system for inheritance** (we'll return to this after we have seen a bit of Python)

Data-directed programming in R

To clarify things a bit, **a class specifies how objects are to be represented in computer code**, or, in short, the properties an object must have if it is of a given class -- An object is the instance of one and only one class and we say that objects differ depending on their “state”

New **classes can extend old ones in a process known as inheritance**; the extension might involve new data, for example, or maybe combining two existing classes -- If class A extends class B, we say that A is a superclass of B and that B is a subclass of A

A **method is a function that is invoked depending on the class of one or more of its arguments**, a process we have been referring to as **dispatch**; think back to the way in which `plot()` functioned differently

S3 classes and methods

We'll start with the relatively simple S3 class construction; primarily because there's not much to it and because **it is a big part of the modeling tools you have been using** in other classes

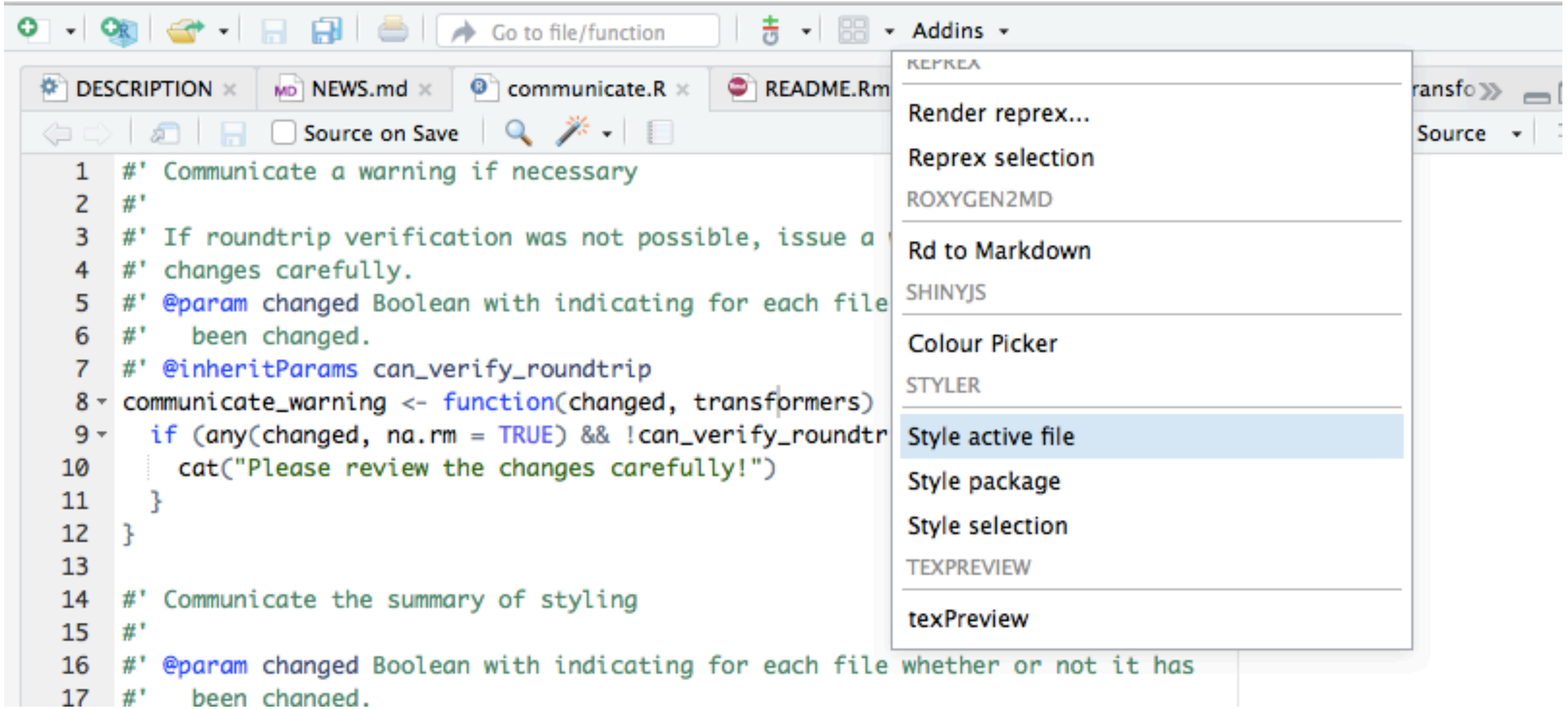
There is no formal mechanism for representing instances of a class; they are, however, typically lists, where named elements of the list represent "slots"

You can access the class of an S3 object with the function `class()`; this can be used to both determine as well as set the class of an object (except for special cases involving implicit classes, this is the same as creating an attribute called `class` with value the string with the class name); finally `is.object()` tests to see if an R object has a `class` attribute

Code style

- https://style.tidyverse.org/

Your turn— style some code?



Random numbers

Famous Quotes

- "The generation of random numbers is too important to be left to chance."
  – Robert R. Coveyou

- "Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin."
  – John von Neumann

06750	40432 33249	53985 30223	99287 52542	72912 71573	27159 25506
06751	48600 14984	50460 06978	25651 97125	16982 90381	96453 15722
06752	11189 98129	91287 82537	90248 10023	56746 24623	31181 80058
06753	82316 89002	53626 21874	61847 04496	67587 12189	58483 91491
06754	20997 65043	19874 76083	89039 48212	54892 47055	09481 44984
06755	24868 96203	07446 98785	90568 25505	10526 23537	25312 62816
06756	78648 71527	12128 43803	16018 16469	92482 04983	53133 76609
06757	01582 09355	52659 58010	56912 77086	50164 20066	03033 60328
06758	62042 15129	68861 69705	03278 05591	09380 79136	99776 85437
06759	08952 76364		39860 01057	98245 58755	32656 40209
06760	31237 32621	91884 73853	47087 36389	86074 26221	49873 15734
06761	31028 57629	10483 60319	55888 01759	40195 78213	77972 72633
06762	54547 73653	61245 31210	80410 39481	29668 94090	59069 91895
06763	12422 40248	81006 89635	08350 40869	61353 41854	00599 98146
06764	99540 92572	80528 47006	90136 41888	17425 72056	51519 80601
06765	30085 92052	27539 81547	78065 49154	80614 32645	22883 38408
06766	54768 10191	33487 58048	06931 06446	34136 05200	16627 30587
06767	37492 46786	95895 05658	12747 76745	36801 78810	18451 82331
06768	89205 52190	84478 48627	78878 81016	90237 75868	74138 31049
06769	15562 98178	93768 92268	24055 18545	51795 62616	30585 24928
06770	43381 37806	16691 52867	84321 83465	40413 59342	10225 85335
06771	18179 14789	94383 42371	28489 93004	53677 28898	44325 32571
06772	83082 16929	16647 44438	21322 63172	63264 21953	73674 70856
06773	64213 16717	88091 60883	45369 07242	10120 30736	04284 78621
06774	24810 68101	86650 79566	42871 92969	72307 01124	00641 44702
06775	56980 54483	62382 06004	57984 22092	70100 87659	56582 40347
06776	17696 47872	45484 26901	13451 73439	18105 30875	52358 30048
06777	61877 07645	15918 45203	57170 29080	80885 91882	69552 31836
06778	02536 82145	62292 18862	38012 47524	34204 82412	45104 32418
06779	98342 51123	05400 98251	06674 53307	33783 57348	62806 47774
06780	02652 55557	25685 39051	54410 78284	35735 72568	80939 08575
06781	70215 12348	52486 72305	31980 32868	58705 77197	36140 45481
06782	87602 88456	85683 47416	55553 08875	26927 73700	92852 78477
06783	84685 96266	90199 77595	52574 89909	76356 64473	60334 90544
06784	17039 40915	56856 16023	05355 36957	85772 90294	75213 83497
06785	30412 93464	23975 02631	08840 26510	31129 66076	90114 62945
06786	29266 56176	87732 80467	21401 76766	45862 81831	88846 80776
06787	87519 34165	00825 43652	17719 84302	66857 01991	33811 46869
06788	57405 27107	48438 59909	99800 22244	14782 08766	49465 05350
06789	27585 04368	67020 36268	36506 98213	72484 55329	78987 56724
06790	11997 08214	93228 99860	62997 97383	80919 76764	69087 93475
06791	30213 55553	74372 48850	86751 02979	29212 62519	51702 50296
06792	50739 96291	57363 46271	84866 60839	15590 13072	42023 64080
06793	47922 50709	38179 28103	66983 10368	41073 74074	13511 37505
06794	72146 89466	66108 25880	34674 39968	56710 48873	16957 96974
06795	48415 90116	26027 99006	35727 76348	09565 28933	19189 75033
06796	22904 19350	96318 85926	51104 62840	45496 69171	31263 67161
06797	08607 57155	85505 37632	71377 67331	37900 87251	18500 04818
06798	42442 89072	07211 25791	15669 47073	26593 44571	92787 88123
06799	89906 69880	21229 66015	24505 21852	75919 05531	72493 06954

06800	31827 80191	43585 20270	74538 48961	90652 02750	82718 27982
06801	92204 68347	84735 32061	47876 42152	89344 82827	44440 91944
06802	72608 47319	85449 66261	38104 76120	86100 88843	17467 79989
06803	71187 34112	21904 22894	45802 68260	57676 37401	50289 46941
06804	30238 58381	06203 10810	07684 84061	78870 19046	94038 74214
06805	97806 63153	46986 88849	10526 23537	25312 62816
06806	68901 15231	70325 84459	26772 51691	92482 04983	53133 76609
06807	51517 35148	82482 85693	74210 33859	50164 20066	03033 60328
06808	86035 69002	34342 01936	34742 79214	09380 79136	99776 85437
06809	40704 12590	78982 10013	91700 87950	98245 58755	32656 40209
06810	99130 52082	16513 04318	72214 98454	63763 75478	24327 74597
06811	71335 76694	81252 45676	44844 62677	53851 92644	60732 82781
06812	13116 26616	14165 91983	62672 77020	33251 77645	65312 20038
06813	97727 69784	70411 30598	19943 51088	33249 54013	76240 99180
06814	59499 59891	93900 73882	82133 74098	05019 92851	23968 39257
06815	68114 63784	03503 02342	25113 59288	43088 23301	32577 52791
06816	10044 70253	87979 40870	33585 78067	80614 32645	22883 38408
06817	63563 42705	55463 28808	51988 92913	34136 05200	16627 30587
06818	50696 67283	43473 16223	32894 93355	36801 78810	18451 82331
06819	38518 61790	07851 50846	06090 37824	90237 75868	74138 31049
06820	29835 05742	96027 41131	59824 61794	51795 62616	30585 24928
06821	81723 66318	25983 03825	44163 56513	38329 18693	74317 87486
06822	76493 58045	96750 07129	65327 00154	17119 69346	05420 06509
06823	49335 20556	69836 18227	28694 35174	32181 36676	88628 92081
06824	32626 25925	16717 87974	50454 68776	91039 09874	52959 79355
06825	31413 49624	17412 92485	58254 09435	00591 81476	16945 70376	85605 17066	49552 43121	83541 54640
06826	30882 36088	10376 15157	23479 91796	45279 49740
06827	41294 09786	32189 23352	72568 43418	08852 98101	43943 44458
06828	17888 24568	43374 48671	62219 17537	42922 91977	57528 49202
06829	84534 85828	24040 62091	52814 00827	23836 10865	84795 21522
06830	84770 38718	43464 28531	38813 37041	52031 92065
06831	57412 03967	67914 47176	51519 98046	26105 98067	75599 05821
06832	64820 40172	01491 06483	77597 98660	53675 83472	08001 75477
06833	78411 41321	57763 52366	17601 86795	48441 79485	38804 89016
06834	52345 55303	85463 36129	06071 32907	65380 31382	38259 13439
06835	89804 07019	11723 27044	92052 58033	91461 13664	56921 22004
06836	79283 35627	78392 14701	91403 04809	38411 56870	03970 31461
06837	48082 88464	43008 32795	64037 26769	21026 82401	36774 88633
06838	76037 32852	87414 96027	31584 98842	23352 88054	24483 83679
06839	17517 48860	09293 41303	98954 42026	80580 53418	71767 88077
06840	83388 12308	91115 21707	06117 13912	46878 38007	08537 27853
06841	55718 98720	72750 18190	13677 90780	32243 09085	21672 39203
06842	24435 18058	05772 72182	51068 70429	34917 50915	80410 87268
06843	54699 57233	62385 34763	34936 62984	78098 00540	12552 72151
06844	10678 53085	81841 14499	55021 47298	60832 32585	42662 00153
06845	59080 53378	01676 67807	40856 34583	60072 28819	03728 72342
06846	44014 53930	28617 75063	03084 19757	53934 80027	44152 21203
06847	31995 38890	35776 76418	82315 92855	00403 22571	27823 38423
06848	75524 91815	79153 32915	62438 17011	41658 36450	38343 31067
06849	68239 32427	42908 76396	41471 14944	69944 17231	15907 49228
			31568 38087	85315 14236	46858 90079

Random Number Generators (RNGs)

- RNGs produce a stream of random numbers U1, U2, U3, ...
- Usually, U1, U2, U3, ... ~ U[0,1]
- U1, U2, U3, ... are independent

- truly random vs pseudo random

Truly Random Number Generators

- e.g. monitoring of physical processes
- example: www.random.org
- speed?
- reproducibility?

Your Turn

- Go to www.random.org
- Find out, how random numbers are "generated" there.
- Draw 5 random integer numbers between 1 and 100.
- What is the speed of the service? cost? reproducibility?

Pseudo-Random Number Generators

- Pre-determined function that produce “random looking” numbers
- Speed?
- Reproducibility?

Generating random numbers in R

- `runif` (uniform), `rpois` (poisson), `rnorm` (normal), `rbinom` (binomial), `rgamma` (gamma), `rbeta` (beta)
- First argument for all is `n`, number of samples to generate
- Then parameters of the distribution
  (always check that the distribution is parameterized the way you expect)

Simulation— randomization and the bootstrap

difference of 3/4”

flickr: bareego

flickr: duncan

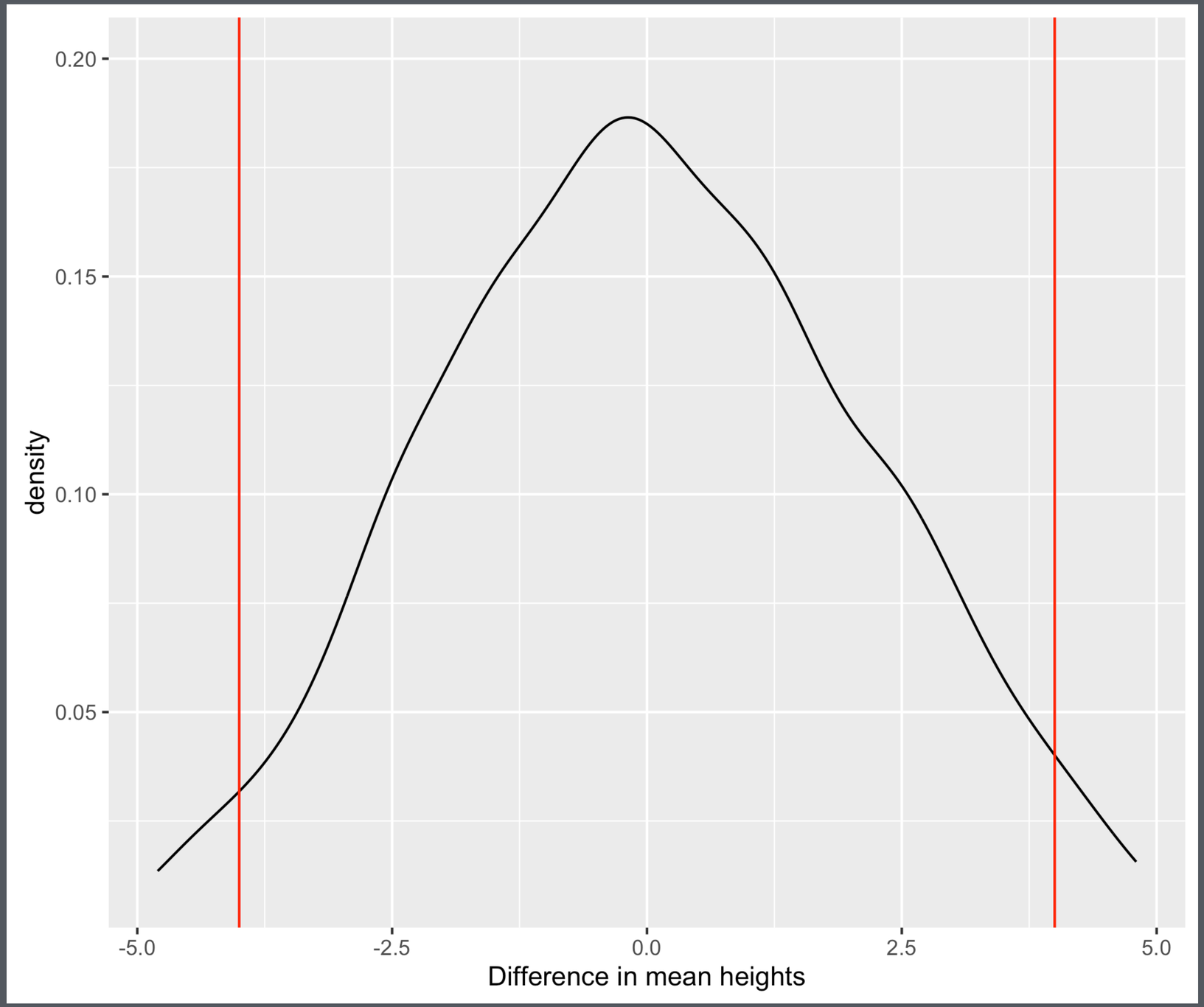
difference of 3/4”

flickr: illinoisspringfield

flickr: illinoisspringfield

67”	blue
66”	blue
68”	blue
65”	blue
73”	blue
68”	pink
63”	pink
69”	pink
71”	pink
72”	pink

bl.ocks: bycoffee

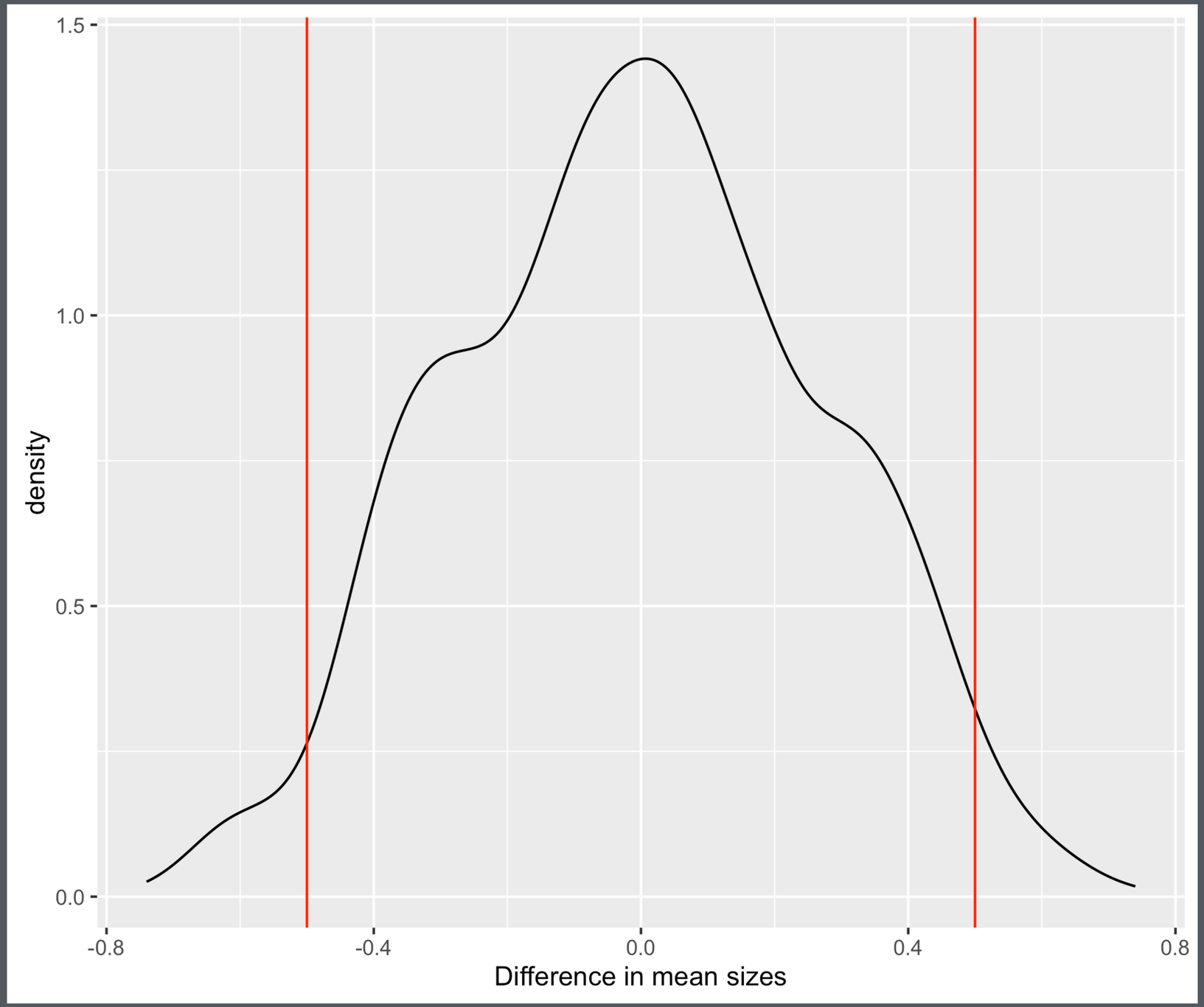

0.20
0.15
0.10
0.05
density
-5.0
-2.5
0.0
2.5
5.0
Difference in mean heights

difference of 3/4”

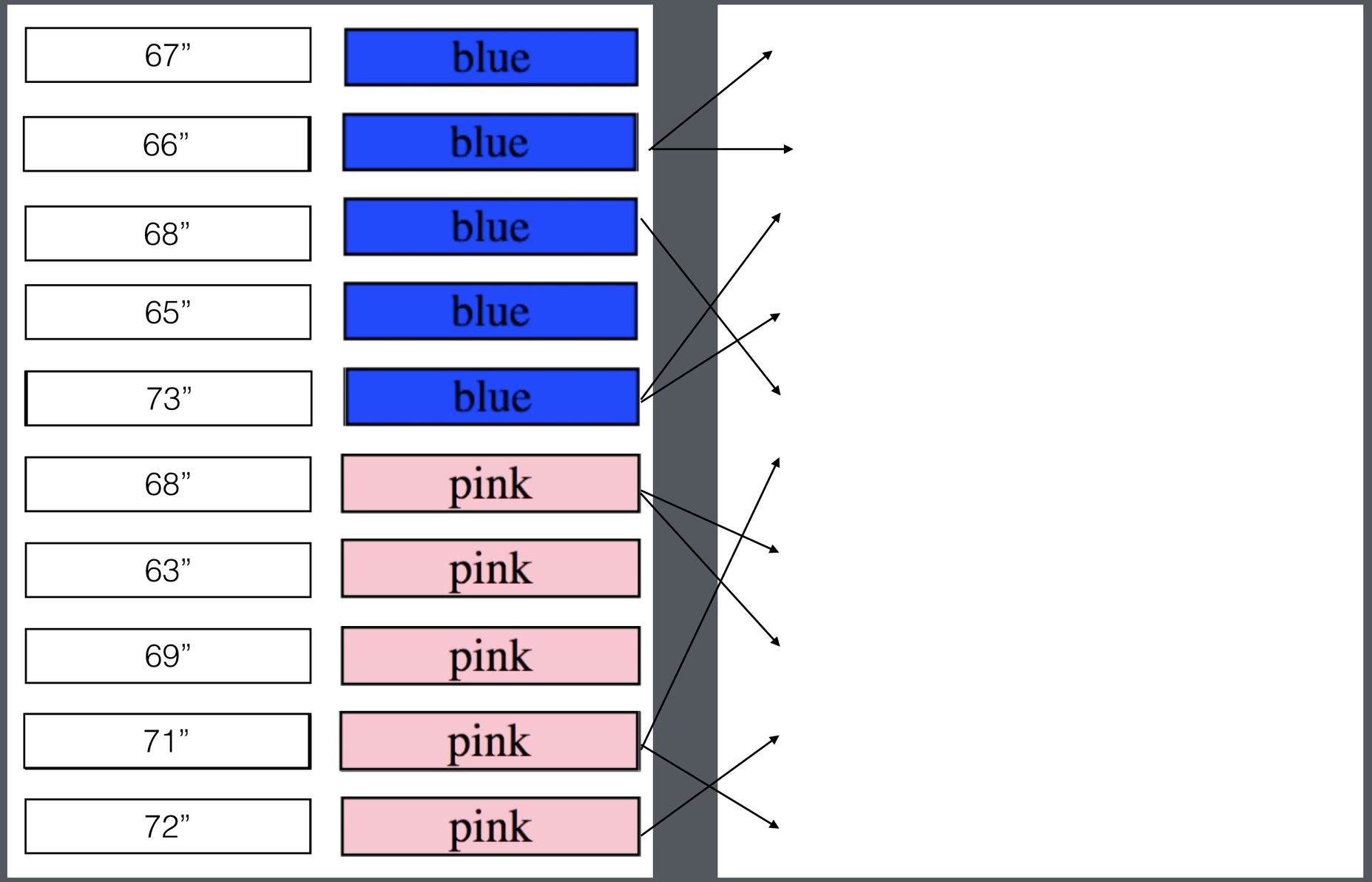
flickr: illinoisspringfield

flickr: illinoisspringfield

(-4,4)


1.5
1.0
0.5
0.0
density
-0.8
-0.4
0.0
0.4
0.8
Difference in mean sizes

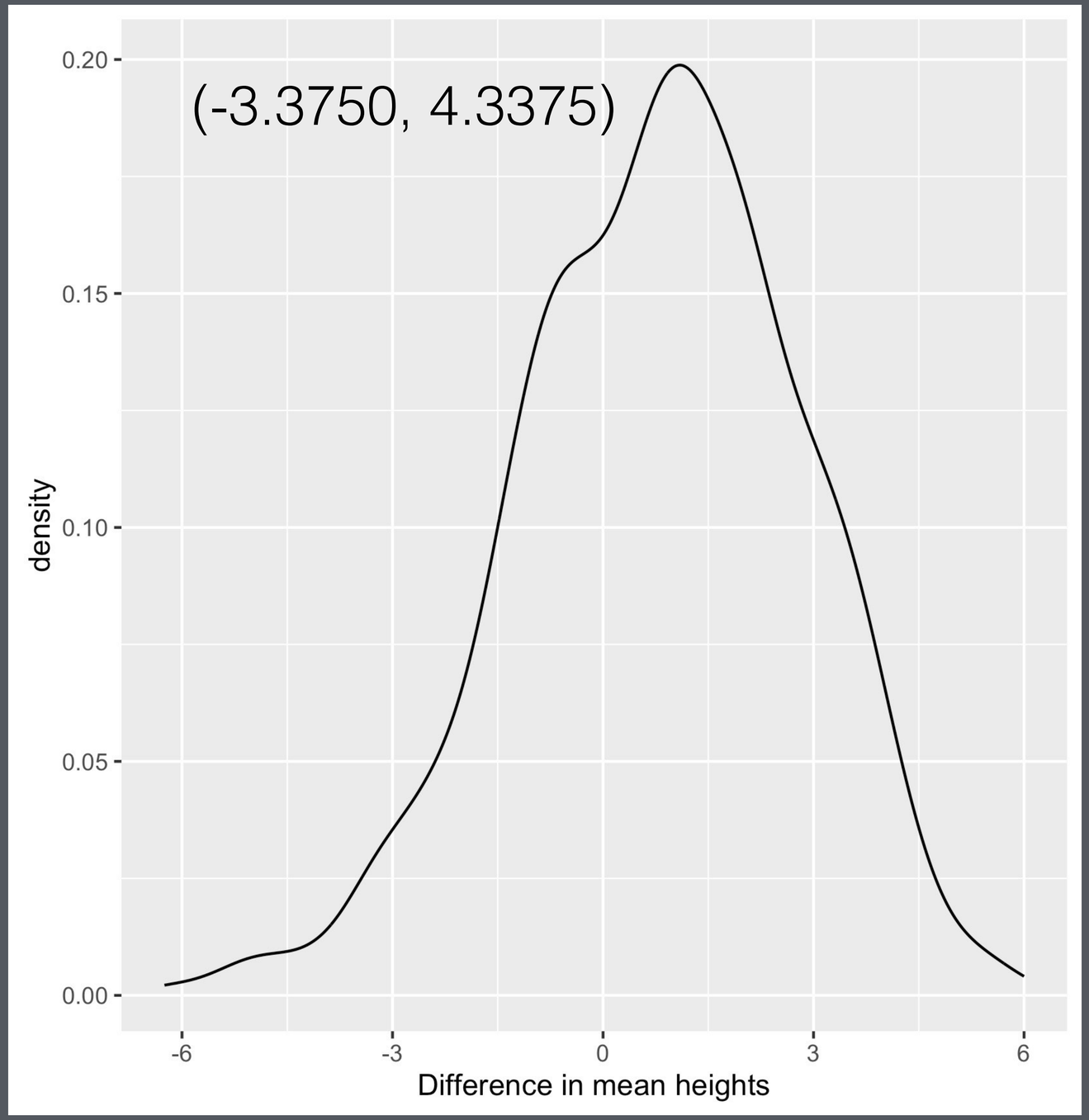
(-0.5,0.5)

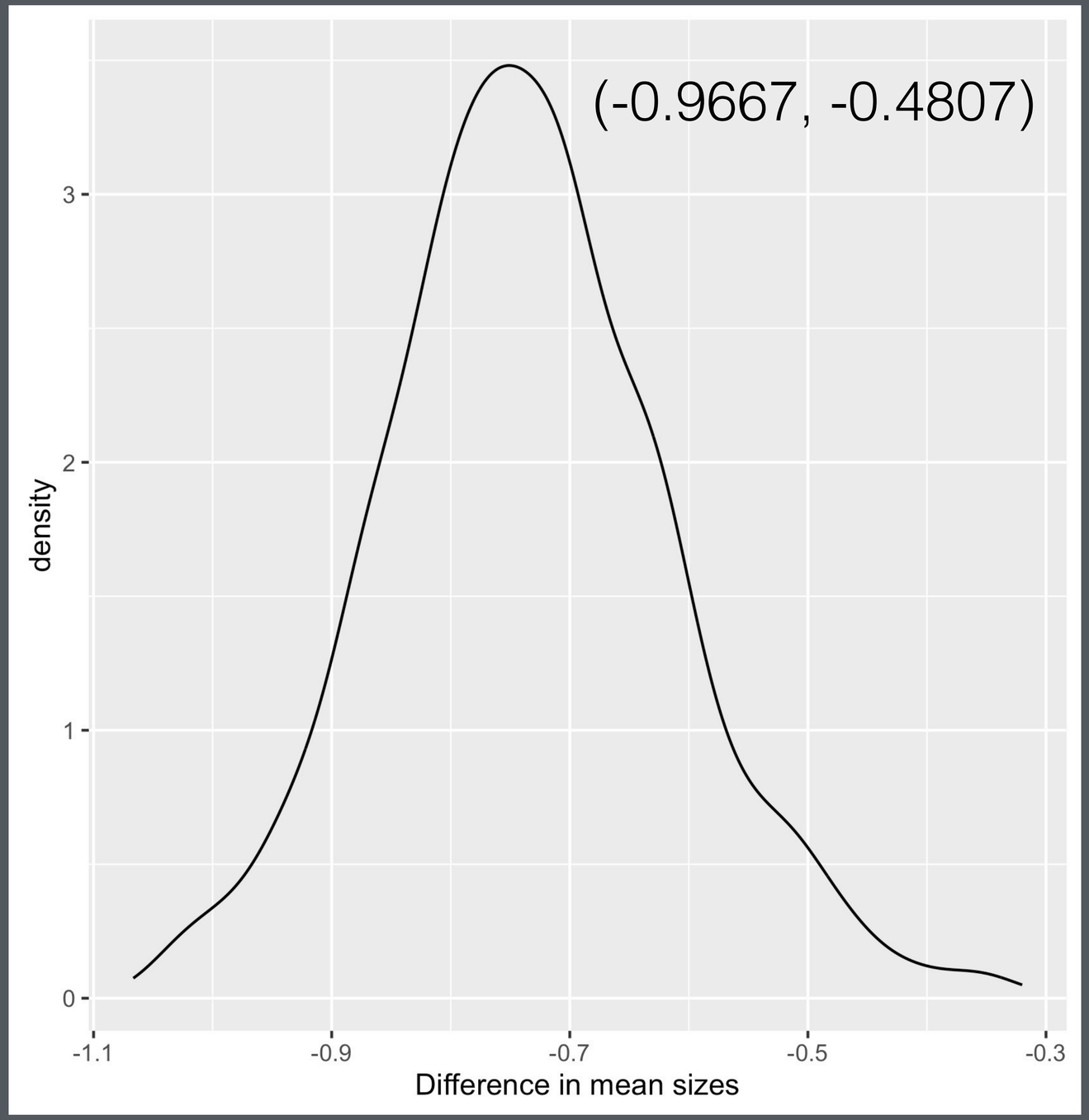

67”
blue
66”
blue
68”
blue
65”
blue
73”
blue
68”
pink
63”
pink
69”
pink
71”
pink
72”
pink

66”	blue
66”	blue
73”	blue
73”	blue
68”	blue
71”	pink
68”	pink
68”	pink
72”	pink
71”	pink

69.2 70

0.8


(-3.3750, 4.3375)
0.20
0.15
0.10
0.05
0.00
density
-6
-3
0
3
6
Difference in mean heights


(-0.9667, -0.4807)
density
3
2
1
0
-1.1
-0.9
-0.7
-0.5
-0.3
Difference in mean sizes