Plotting With R (Vol 1)

Plotting I

Paul E. Johnson[1,2]

[1]Department of Political Science
University of Kansas

[2]Center for Research Methods and Data Analysis
University of Kansas

2013

Outline

Outline

R's plot is Magic

- Do you have a data frame "mydata"? Try this:

```
plot(mydata)
```

- Do you have 2 variables, x and y inside mydata? Try this:

```
plot(y ~ x, data = mydata)
```

- I prefer that "formula interface," but it is also going to work if you run this:

```
plot(mydata$x, mydata$y)
```

  Or,

```
with(mydata, plot(x, y))
```

R's plot is Really Magic

- plot will try to guess what kind of drawing you want

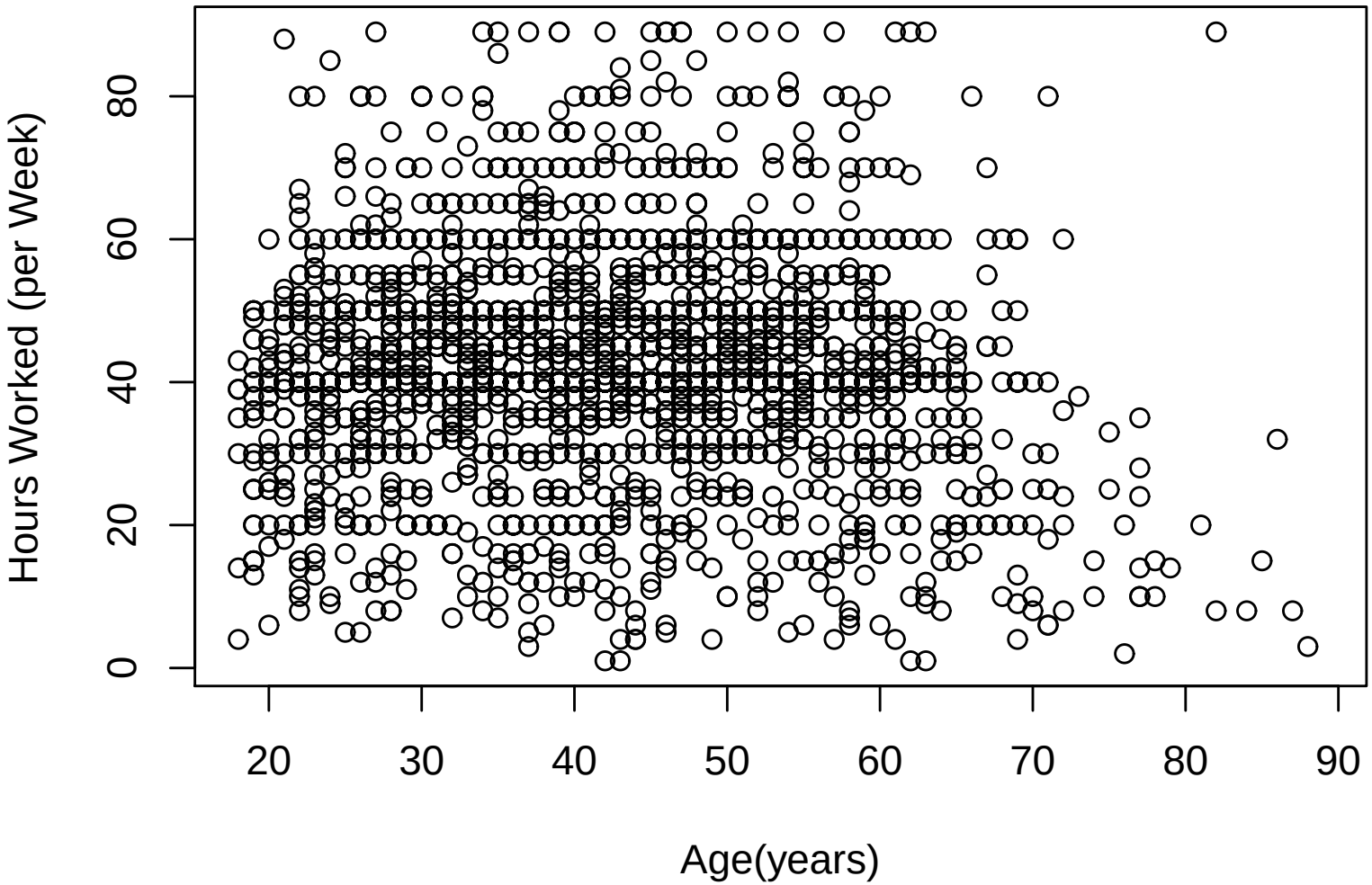
```
plot(y ~ x, data = mydata )
```

  Depending on the characteristics of x and y, this might generate completely different kinds of figures.

- Figure result depends on 2 things.
  1. The primary argument–which implies data types for R to inspect
  2. Default graphical settings (inherited from the larger graphic environment (run "par()" to see it)).
- My usual workflow is like this
  1. use plot with defaults
  2. notice flaws, start removing defaults and replacing
  3. most radical approach is to add arguments

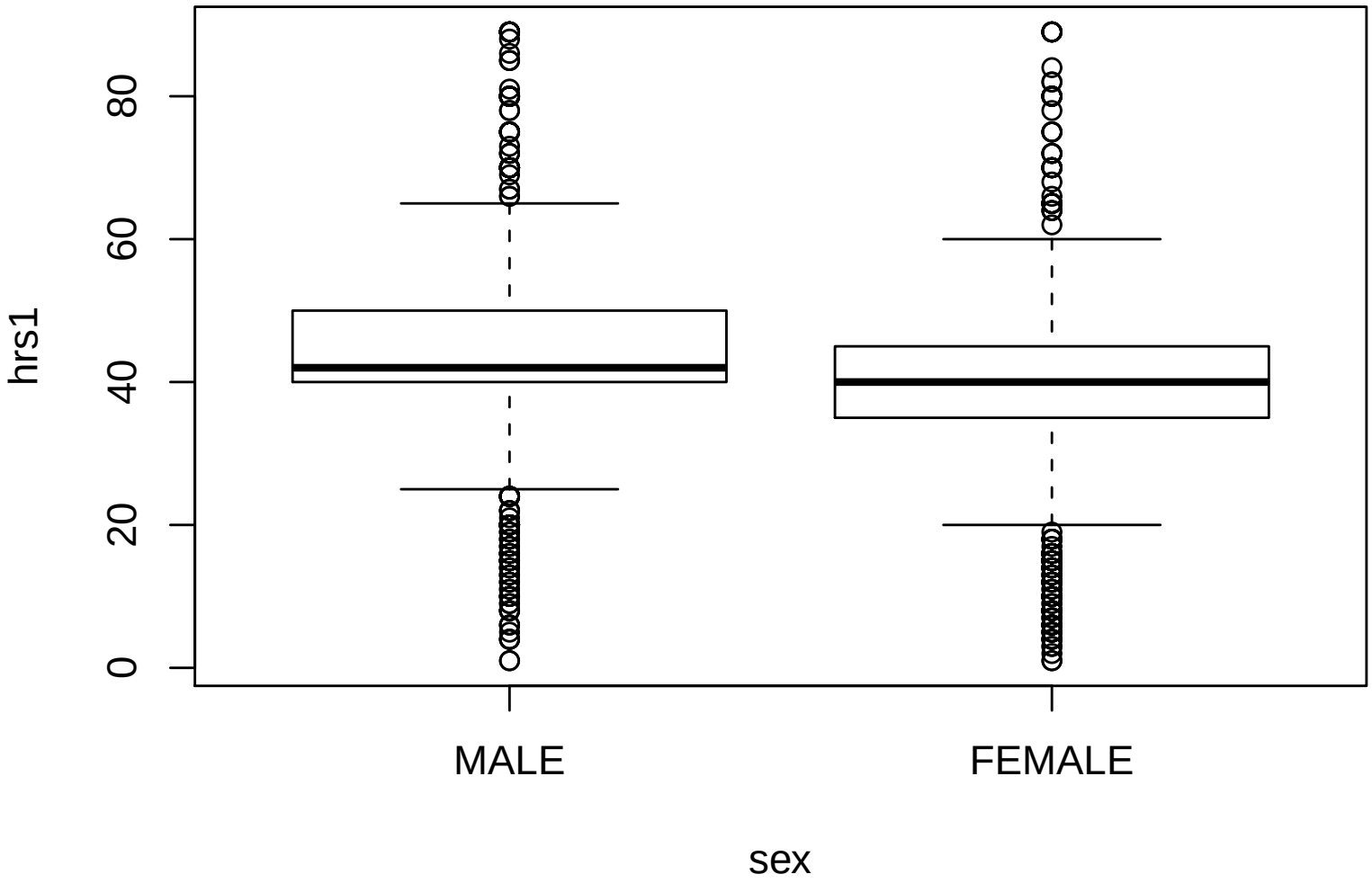
```
## plot( ...whatever..., type="n", axes = F)
```

Scatterplot: y and x are both numeric



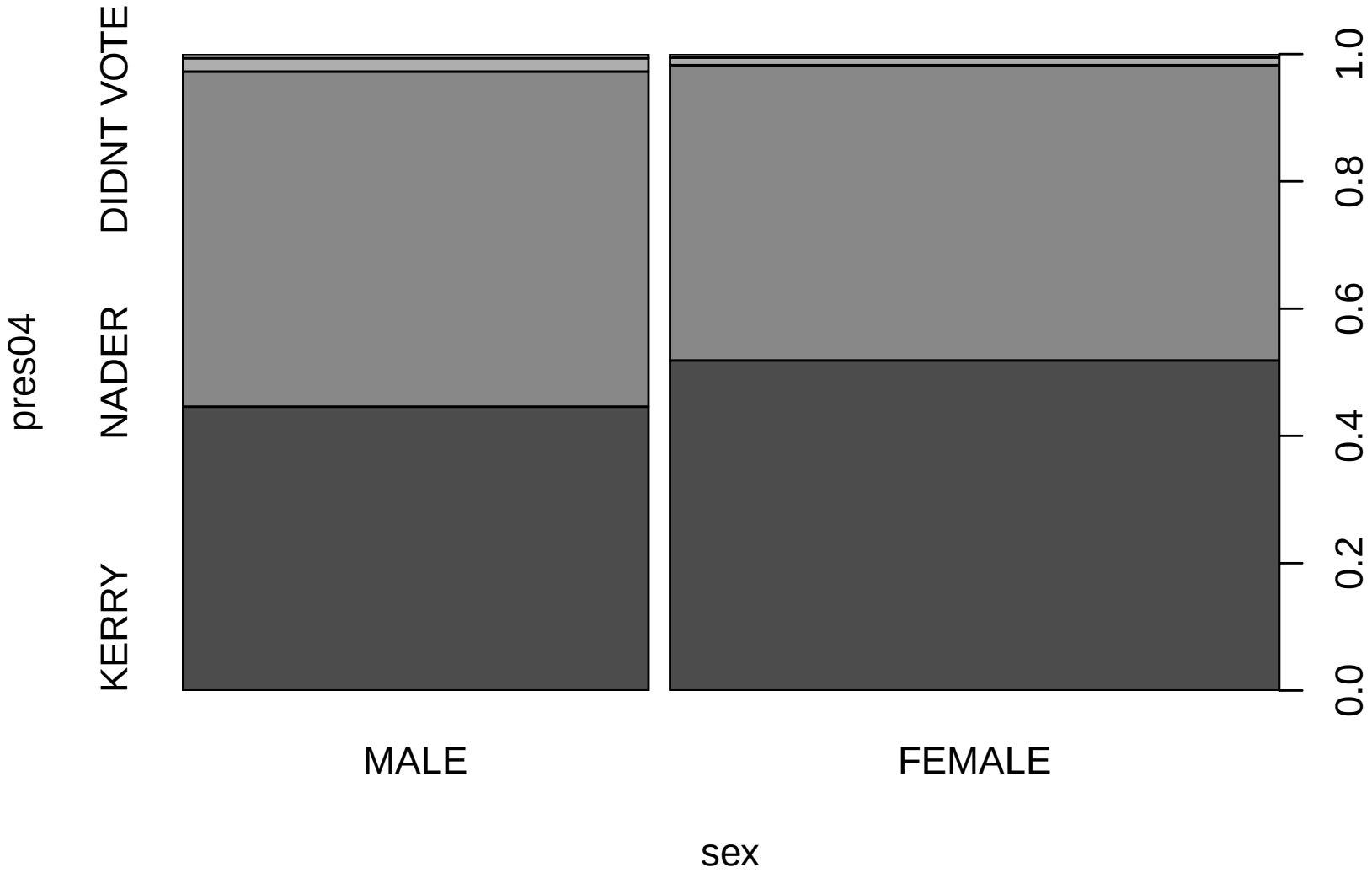
General Social Survey (2006)

"Box" plot: y numeric, x categorical



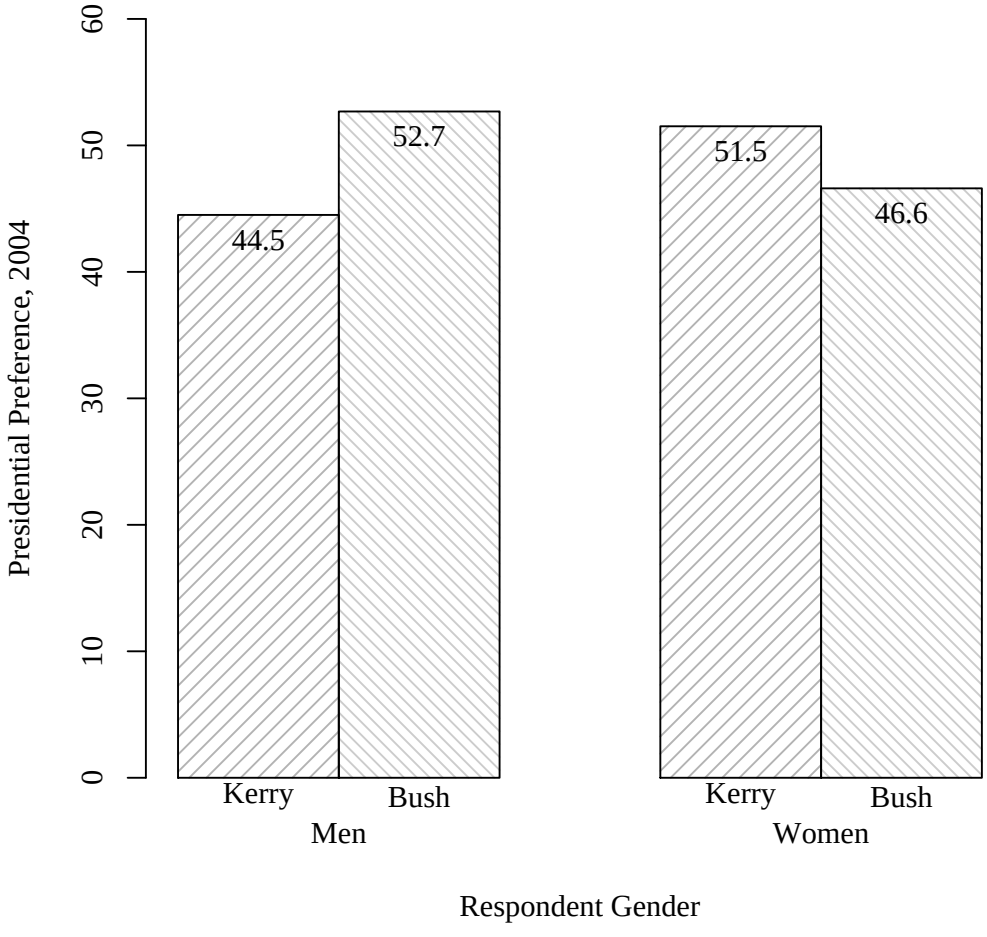
General Social Survey (2006)

"Ugly Barplot": y categorical, x categorical



Actually, this is a "spinogram"

But It Doesn't Have To Stay Ugly



Used the "barplot()" method to get more control

Read "?plot". plot is a "generic function" for 2D drawings

- terminology: **a generic function** is an "entry point" to a family of functions that do special work on objects of various types.
- **method function** Run: `someFunction(some.object)`. The R runtime system asks `some.object` what its class is, and then it looks for a *method function* named `someFunction.class` that will do the work.
- `plot (y ~ x, data=dat)` has a formula as its first argument, so `plot.formula` is called. It discerns the data types and sends work to "barplot", "boxplot", etc.
- This framework simplifies user interaction with R by reducing the number of separate function names that one must learn.
- Not necessary to use generic `plot` to make a barplot. Read `?barplot`, use it directly if you want to. If I know I want bars, I don't mess about with `plot()`.

R also will try to guess what you want from an object

- Most calculations in R create "R objects"

```
#      myObject <- SomeWonderfulFunction (X, myArgument = 7)
```

- You could ask `myObject` for its type

```
#   class(myObject)
```

- If you run

```
#   plot(myObject)
```

R asks for the type and then finds the right method.

What kinds of objects will plot handle? run "methods(plot)"

[1] plot.acf*
[2] plot.data.frame*
[3] plot.Date*
[4] plot.decomposed.ts*
[5] plot.default
[6] plot.dendrogram*
[7] plot.density
[8] plot.ecdf
[9] plot.factor*
[10] plot.formula*
[11] plot.hclust*
[12] plot.histogram*
[13] plot.HoltWinters*
[14] plot.isoreg*
[15] plot.lm
[16] plot.medpolish*
[17] plot.mlm
[18] plot.POSIXct*
[19] plot.POSIXlt*
[20] plot.ppr*
[21] plot.prcomp*
[22] plot.princomp*
[23] plot.profile.nls*
[24] plot.spec
[25] plot.spec.coherency
[26] plot.spec.phase
[27] plot.stepfun
[28] plot.stl*
[29] plot.table*
[30] plot.ts
[31] plot.tskernel*
[32] plot.TukeyHSD

Non-visible functions are asterisked

That's TMI

- Before you skip past it . . .
- notice syntax. "plot.ts" : if you have a "ts" object called "myobject", something is supposed to happen if you run:

```
#  plot(myobject)
```

- If the R runtime finds no method your object's class, it will try to run

```
#  plot.default(myobject)
```

Here's an example with a "density" object.

- Create 1500 random numbers from a Poisson, get the density object, then plot it.

```
myx <- rpois(1500,lambda=17)
den1 <- density(myx)
plot(den1)
```

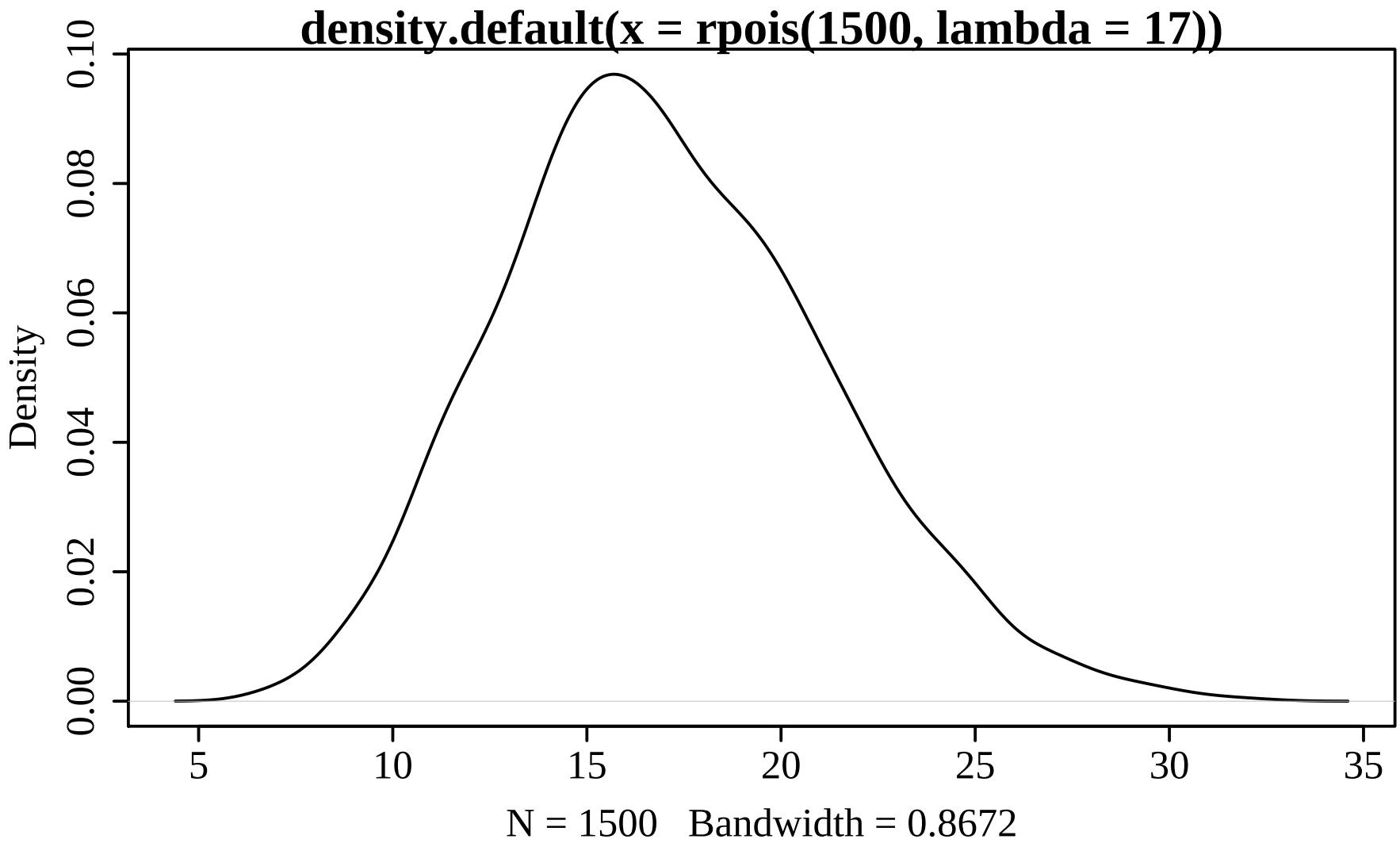
- Same as creating object "on the fly"

```
plot(density(rpois(1500, lambda = 17)))
```

- Except that this leaves behind no object to investigate.

The default plot of the density object, output from

```
plot(density(rpois(1500, lambda = 17)))
```



But I did not want that plot!

- If plot() gives the wrong output, then customize it!
- Investigate "den1", the density object

```
attributes(den1)
```

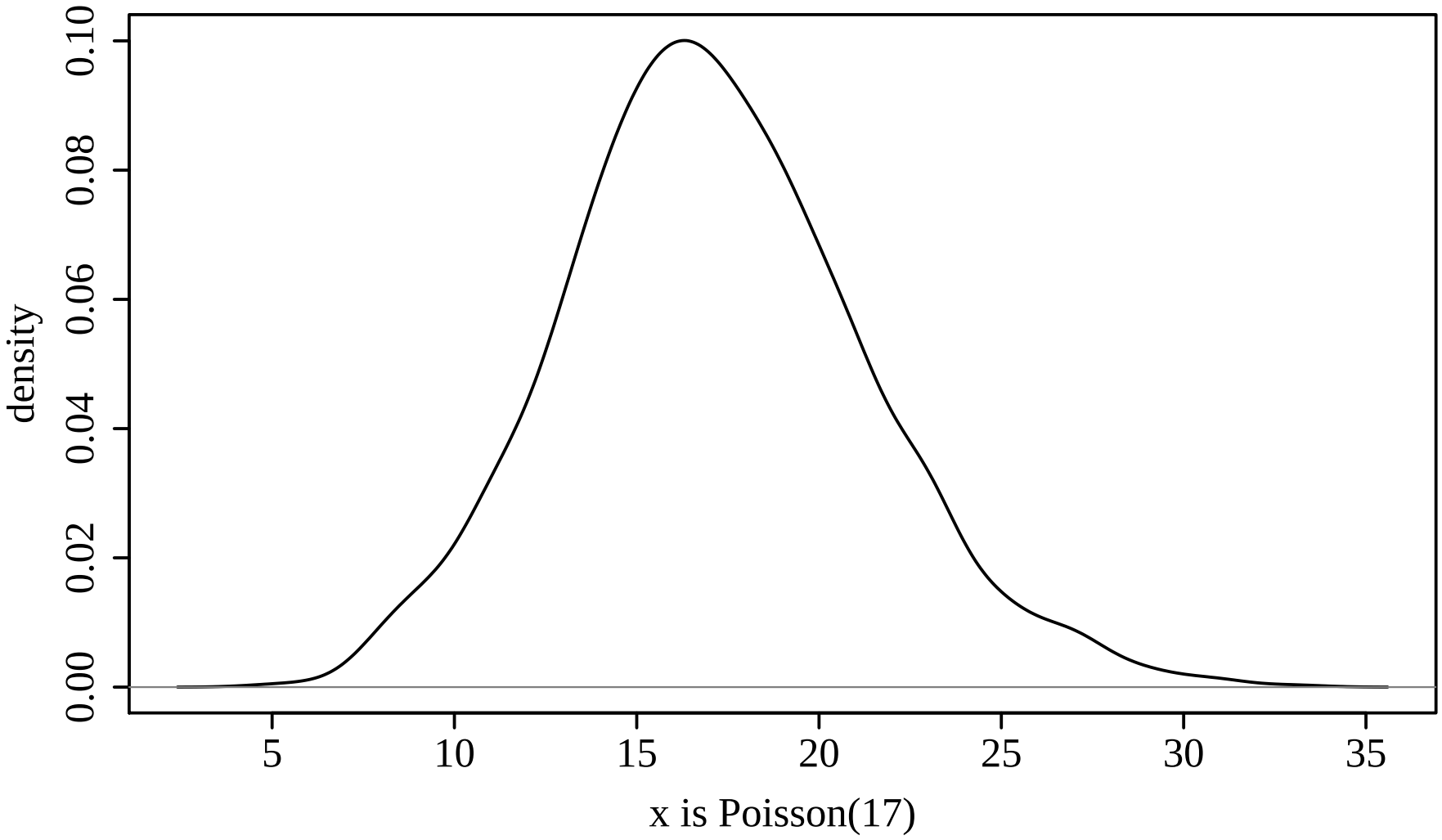
```
$names
[1] "x"         "y"         "bw"        "n"         "call"
    "data.name" "has.na"

$class
[1] "density"
```

But I did not want that plot!

- Use the information inside that object to make the kind of plot you want, using the plot tools described in the next section.

```
plot(den1$x,den1$y,type="l", xlab="x is Poisson(17)", ylab="
    density")
abline(h=0, col="gray50", lwd=0.6)
```

A customized plot of density



Outline

High Level Functions allocate a "device" and draw on it

- High level functions
  - plot.default
  - hist
  - barplot
  - boxplot
  - dotplot
  - piechart
- Ironically, you can ask these functions to not plot!
  - plot(x, y, type="n")
  - That will create the "device" and draw axes only.
  - Ask for no axes: plot(x,y, type="n", axes=F)

Low Level Functions: "doodle" on an Existing Plot Device

- Low level functions
  - text: for writing inside the plot area
  - mtext: for writing margin text
  - points: puts plot characters in plot area
  - abline: draws straight lines
  - lines: draws a smooth curve through indicated points
  - legend: for annotation of lines, points, or bars
  - segments: draws straight lines between indicated points
  - polygon: for a shaded region in the plot area
  - axis: customize axes
  - and others exist
- Once you get the hang of it, you can make some awesome, publication-quality drawings.

Various types of options to 2D plot commands

- Hints to the generic itself to help it know what you want plot(x, y, type="s")
- Options common to all 2D plot methods.

  main a main title
  xlab a label for the horizontal axis
  xlim a vector of axis limits, as in c(0,100)

- Options that are interpreted by the graphic environment

  cex=0.5 sets "character expansion factor"

- Options that are intended for the specific method that is chosen. (If the work is dispatched to the wrong method, an error will result).
- The importance of "..." in definition ?plot. An option that *might be used* at some stage in the process.
- The function "par()" can get or set parameters that affect the way plotting methods work.

Outline

Fiddle Around with a Scatterplot

- The "easy, automatic" plot is almost never what you want.
- With numeric x and y, plot(x,y) sends the data to a method called "plot.default"
- Expert users might work with plot.default directly, most users never need to.

Exercise: I've create a simple example dataset: mydf.txt

```
age <- c( 14, 16, 19, 18, 21, 17, 15, 15)
sex <- c( "M","M","M","M","F","F","F","F")
salary <- c(8.00, 7.55, 20, 9.00, 26.00, 7.55, 5.00, 13.00)
mydf <- data.frame( age, sex, salary)
rm(age, sex, salary) ## clean up workspace
write.table(mydf, file = "examples/mydf.txt", sep = ",", row.names
    = FALSE)
```

- Bring mydf.txt into your session

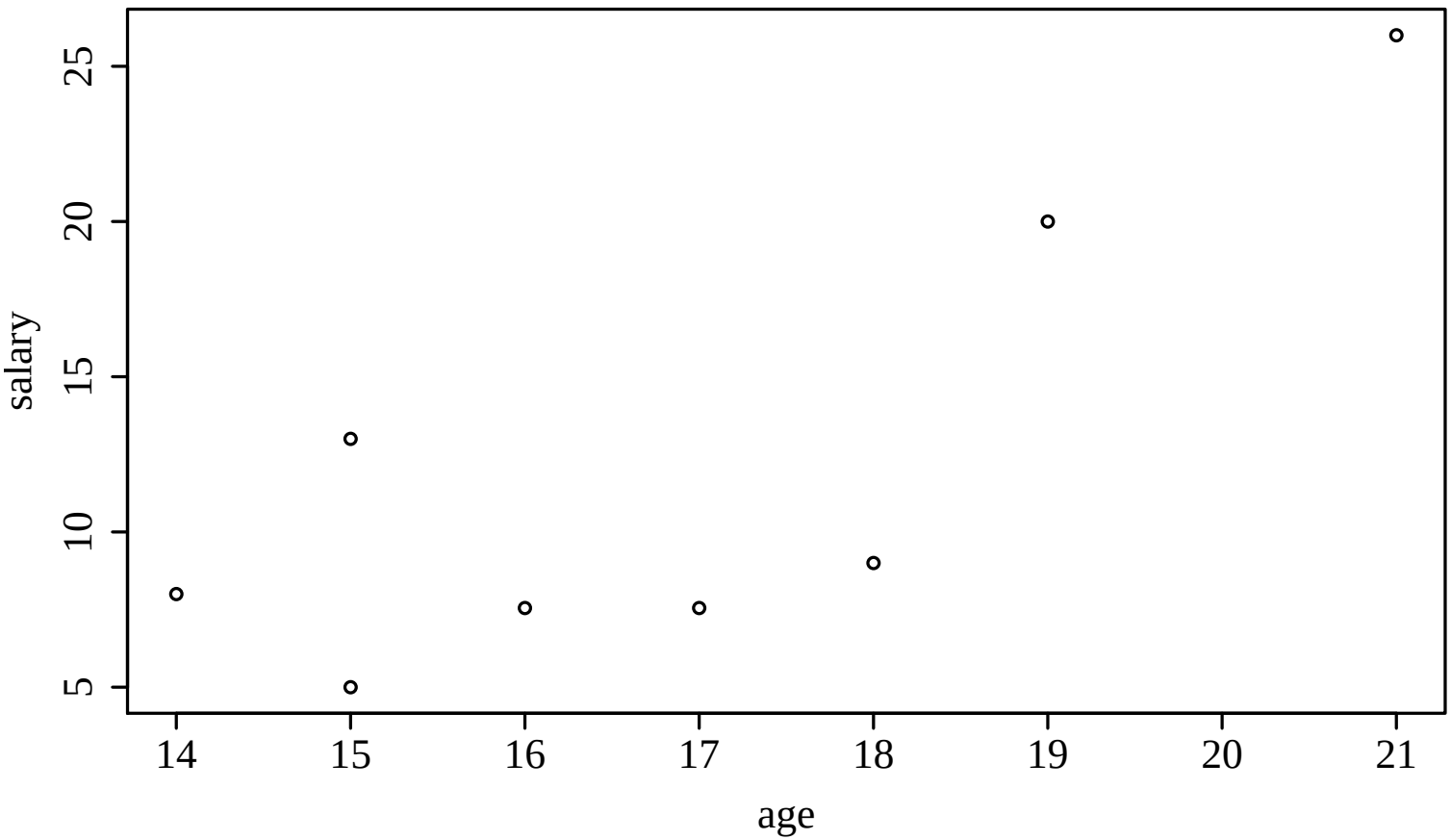
```
mydf <- read.table("examples/mydf.txt", sep = ",", header =
    TRUE)
```

- Test these commands

```
plot(salary ~ age, data = mydf)
plot(salary ~ age, data = mydf, cex = 2)
plot(salary ~ age, data = mydf, col = "pink", cex = 2, pch =
    16)
plot(salary ~ age, data = mydf, col = as.numeric(sex), cex =
    salary, pch = 16)
plot(salary ~ age, data = mydf, type = "l")
plot(salary ~ age, data = mydf, type = "b")
plot(salary ~ age, data = mydf, type = "h")
plot(salary ~ age, data = mydf, type = "s")
plot(salary ~ age, data = mydf, type = "c")
text(salary ~ age, data = mydf, labels = 1:8)
```

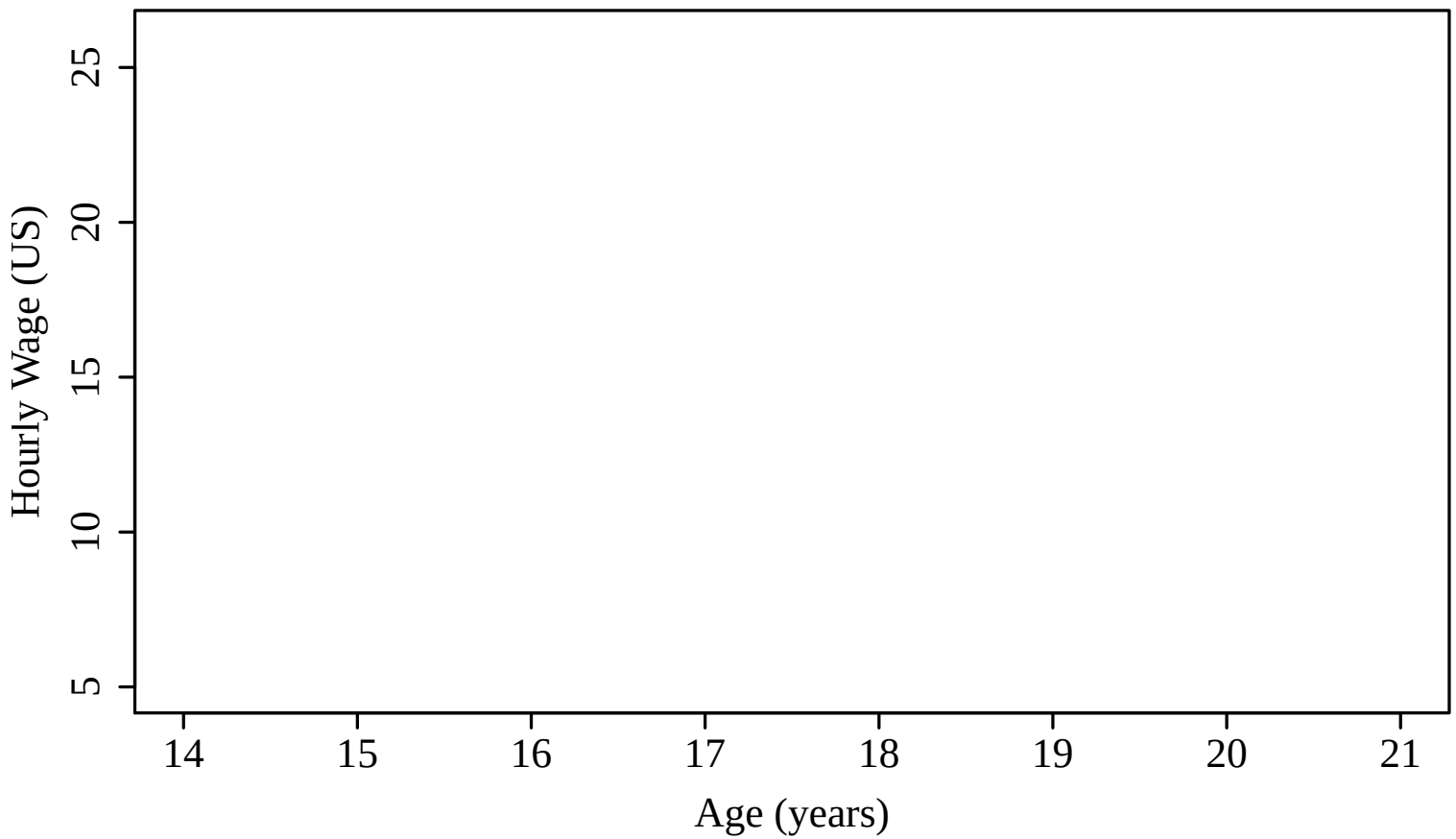
The Default Plot

```
plot(salary ~ age , data = mydf)
```



Make A Plot That's Empty in the Middle

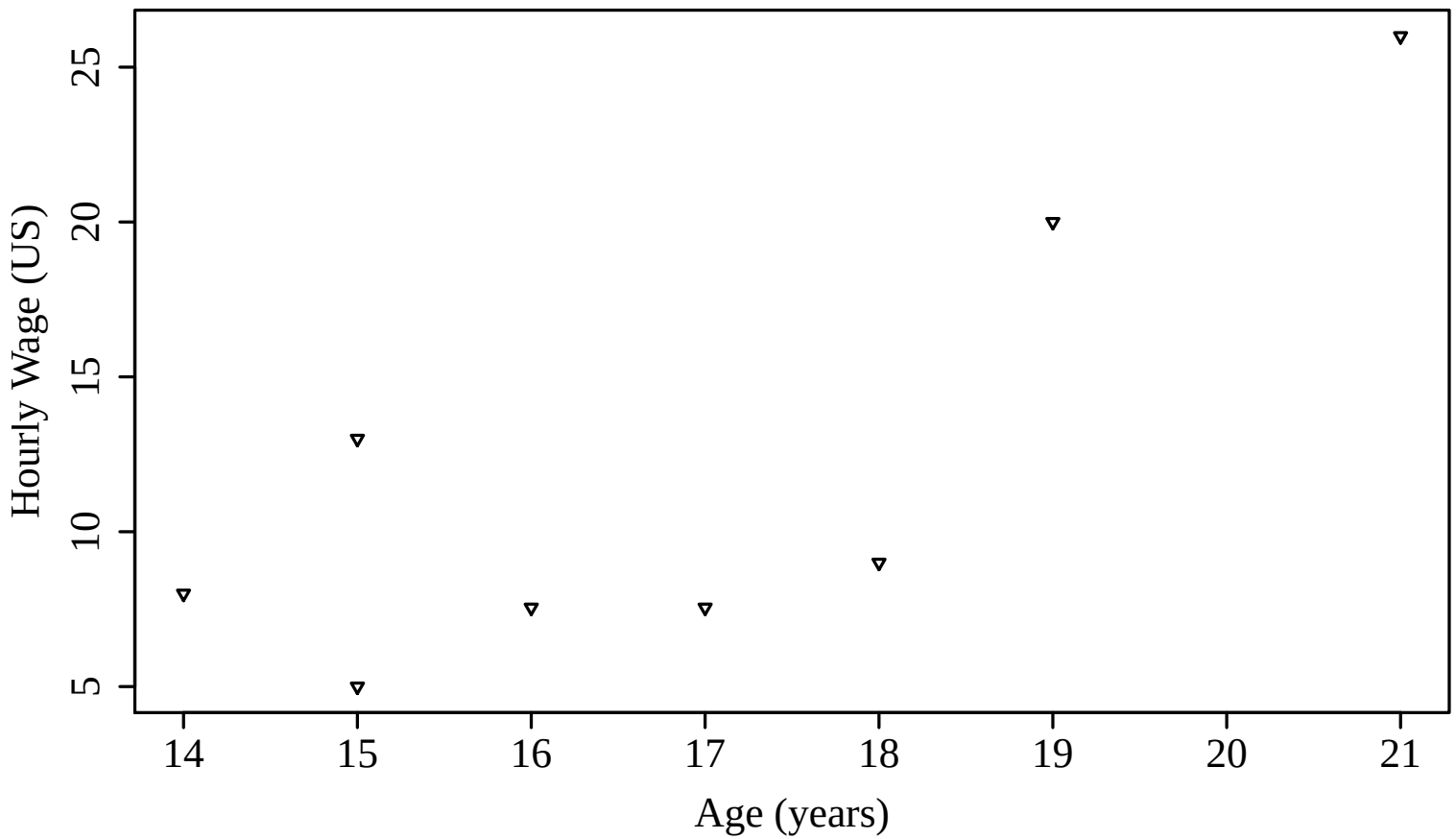
```
plot(salary~age , data=mydf, type="n", xlab="Age (years)", ylab="
    Hourly Wage (US)")
```



Use "points()" to put the points back

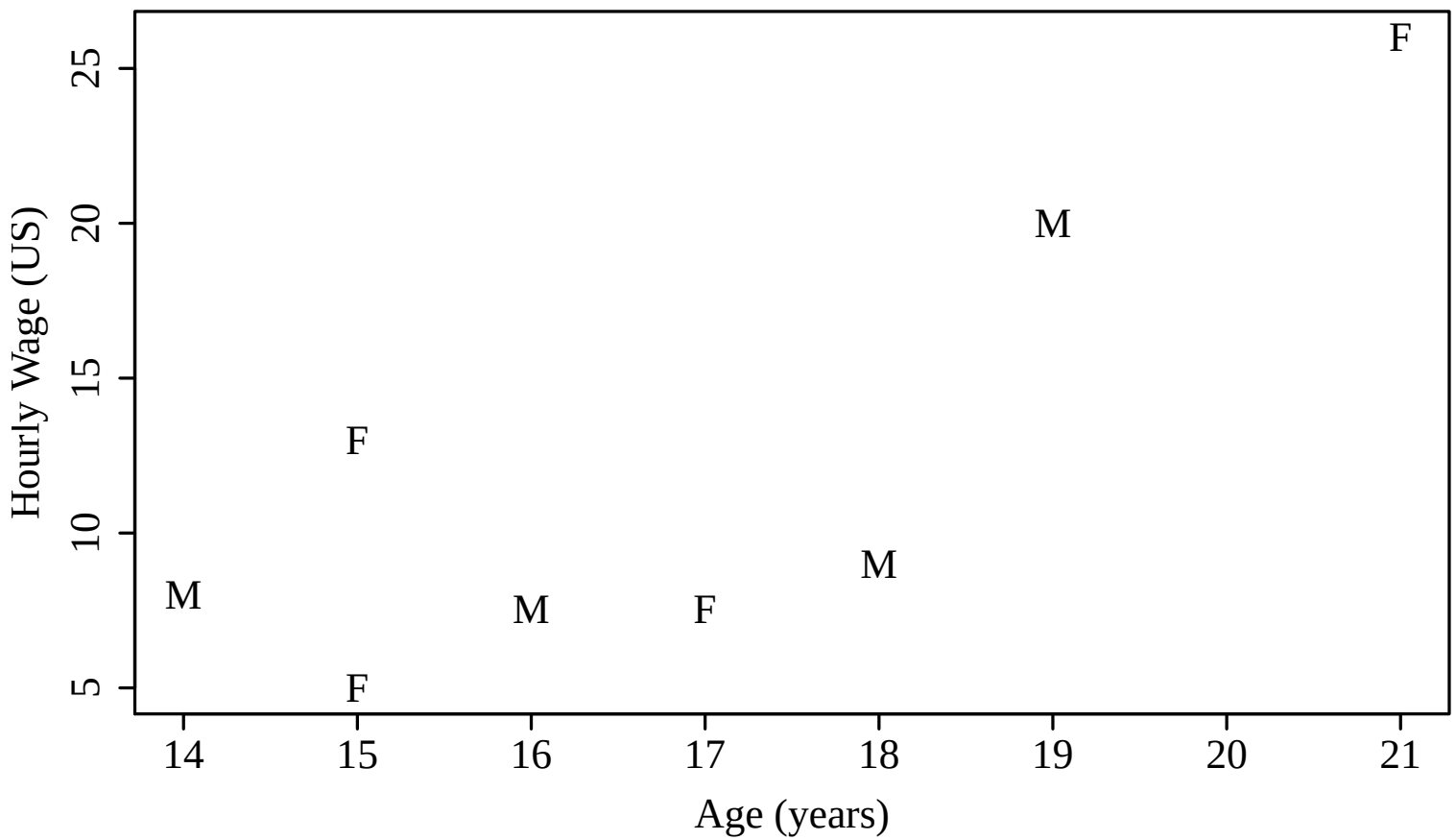
```
with(mydf, points(age, salary, pch=6, cex=0.8))
```

See ?points; pch "plotting character", cex resizes it



Use "text" to write gender data

```
##with(mydf, text(age, salary, labels = mydf$sex)) ## same as:
text(salary ~ age, data = mydf, labels = mydf$sex)
```



Perhaps You Like Named Subjects

- Insert a column of names into the data frame

```
mydf$subjects <- c("Pat","Chris","Kris","Stacey","Leslie","
    Jaime","Terry","Mickey")
```

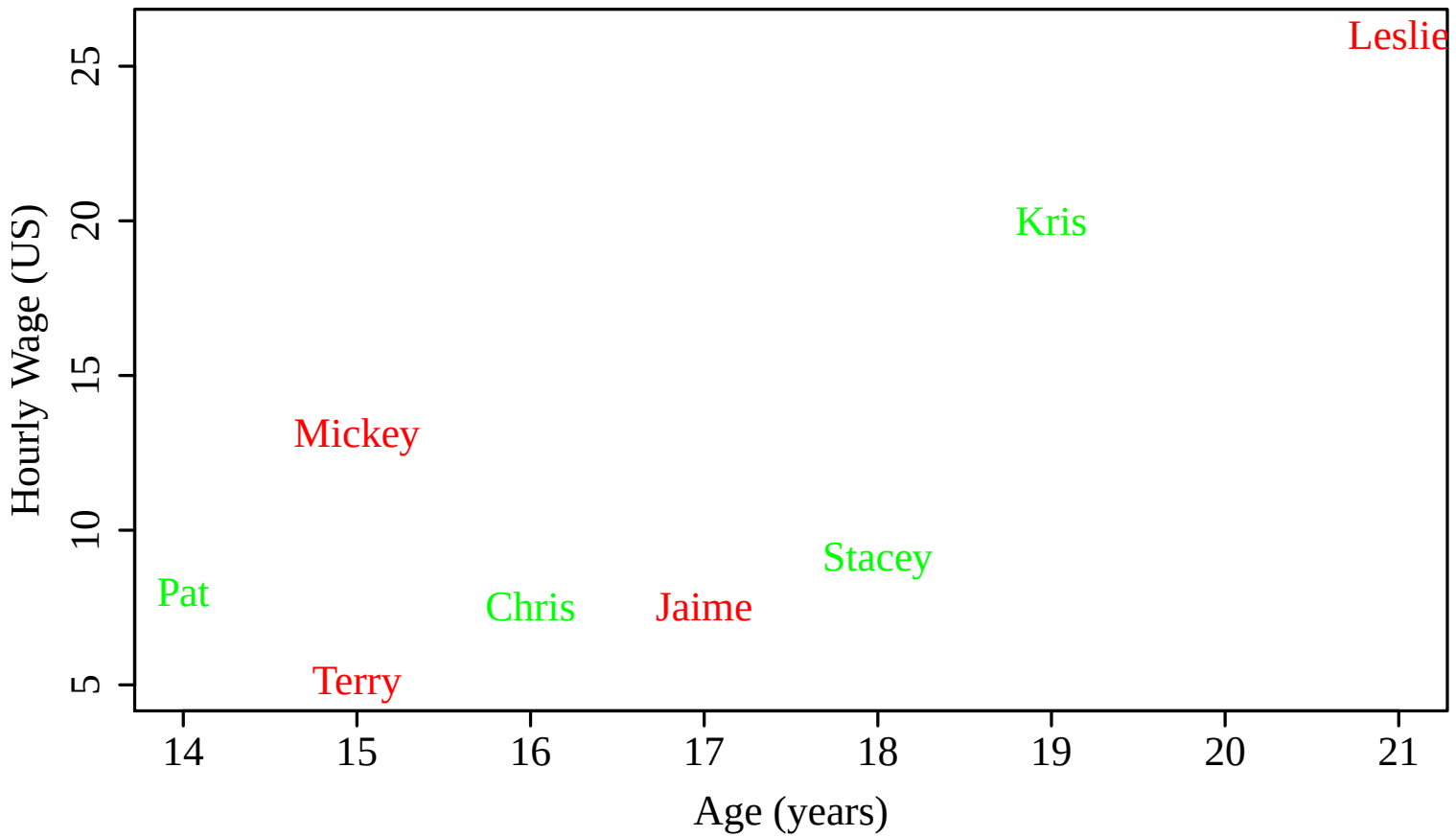
- Redraw the base plot

```
plot(salary~age , data=mydf, type="n", xlab="Age (years)", ylab
    ="Hourly Wage (US)")
```

- Write the names into the plot region

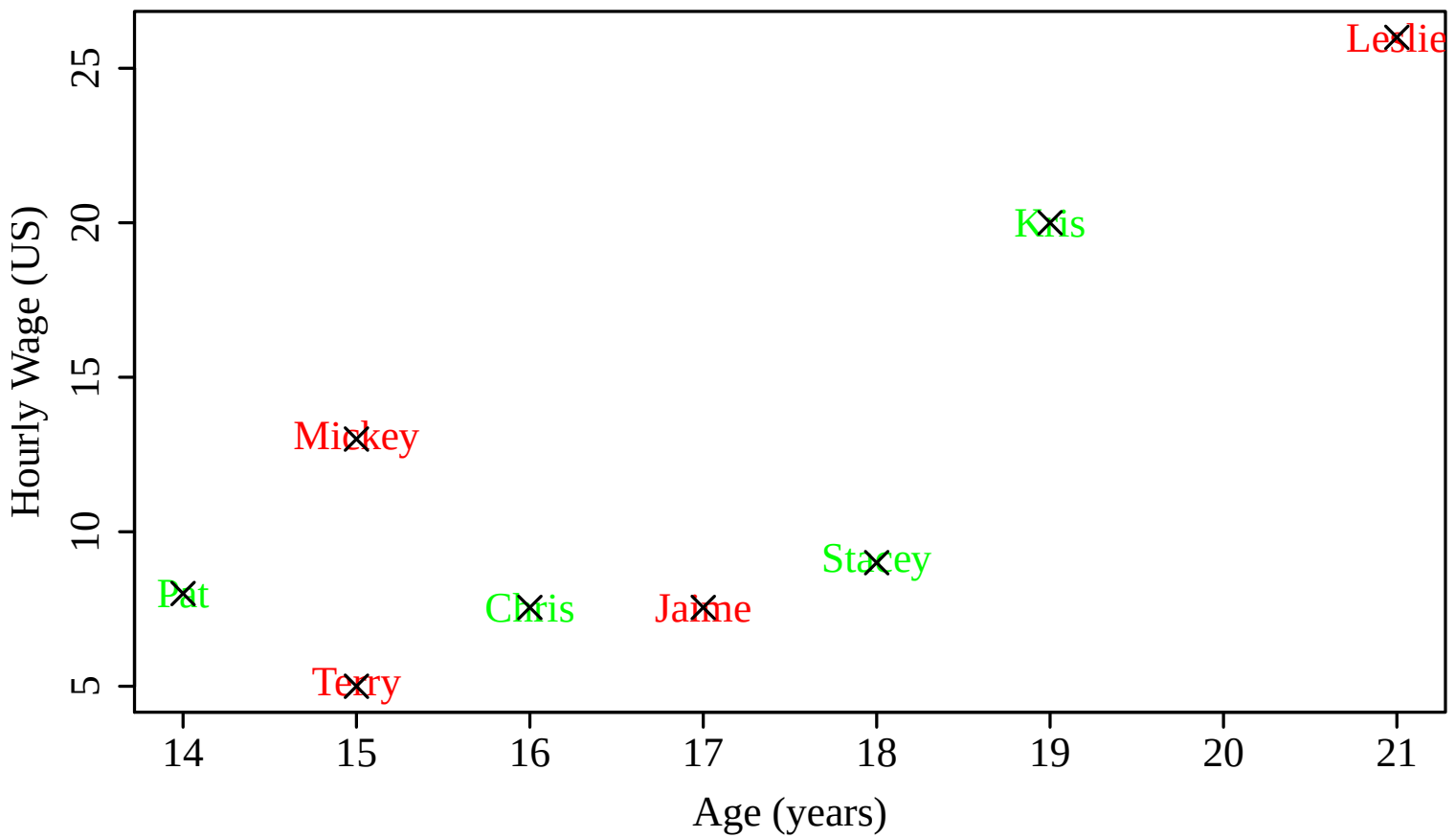
```
myColors <- ifelse(mydf$sex=="M", "green", "red")  ##
    alternating red and green
# with(mydf,  text( age, salary, labels = subjects, col =
    myColors))
text(salary ~ age, data = mydf, labels = subjects, col =
    myColors)
```

Write the names on the plot



Plot Some Giant X's Too!

```
## with(mydf, points(age, salary, pch = 4, cex=2)) ##same as:
points(salary ~ age, data = mydf, pch = 4, cex = 2)
```



Turn Off Everything Visible

Go completely naked! No axes, no box, no nothing.

```
plot(salary~age , data=mydf, type="n", bty="n", axes=F, xlab="",
     ylab="")
```

The Blank Canvas Is So Intimidating

But Its Not *Really* Naked. It is "scaled" drawing area

I'm centered on (16.88,12.01), the means of age and salary

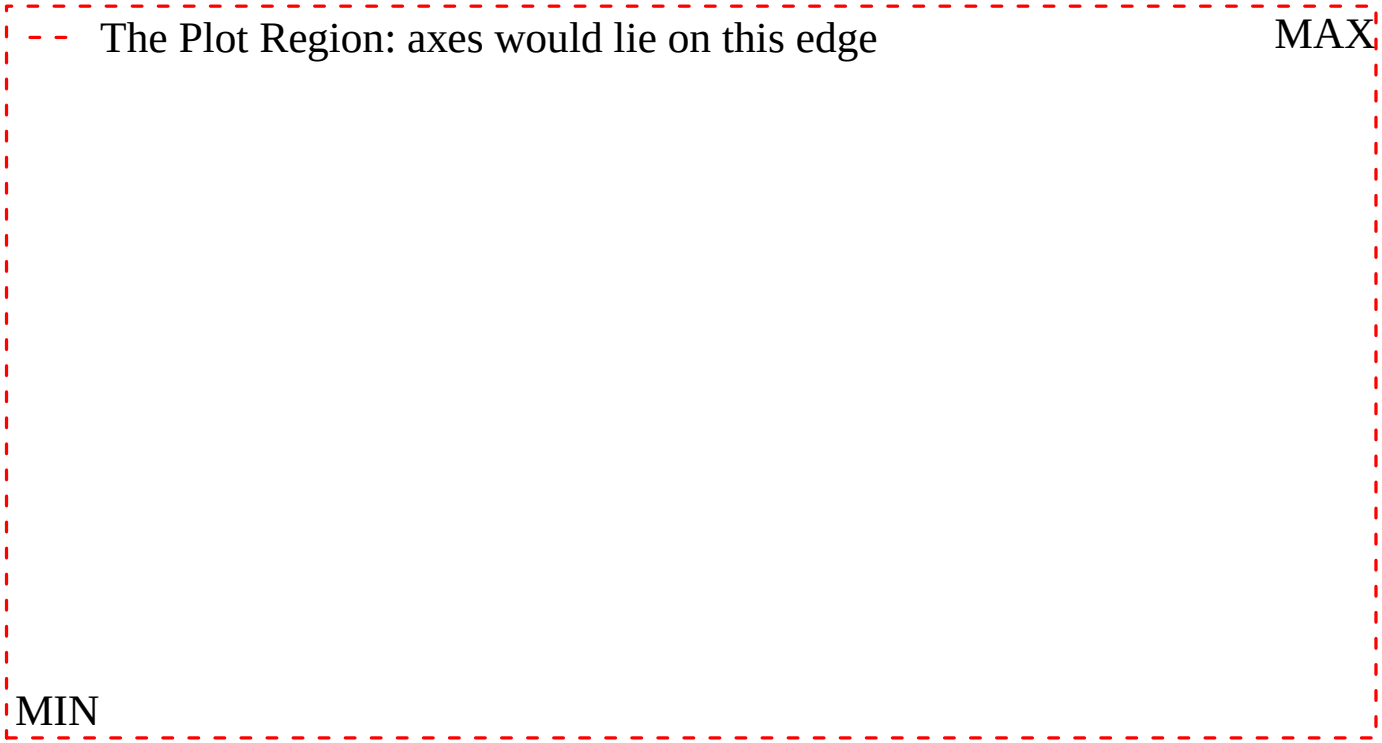
Marking on a Canvas

- Place words “MAX” and “MIN” at the edges of the data.

```
text(max(mydf$age), max(mydf$salary), "MAX", offset=0)
text(min(mydf$age), min(mydf$salary), "MIN", offset=0)
```

- Put a red dotted rectangle for the outer edge of the Plot Region

```
box(which="plot", col="red",lty=2)
legend("topleft", legend=paste("The Plot Region:", "axes would
    lie on this edge"), lty=c(2),col=c("red"), bty="n")
```


The Plot Region: axes would lie on this edge
MAX
MIN

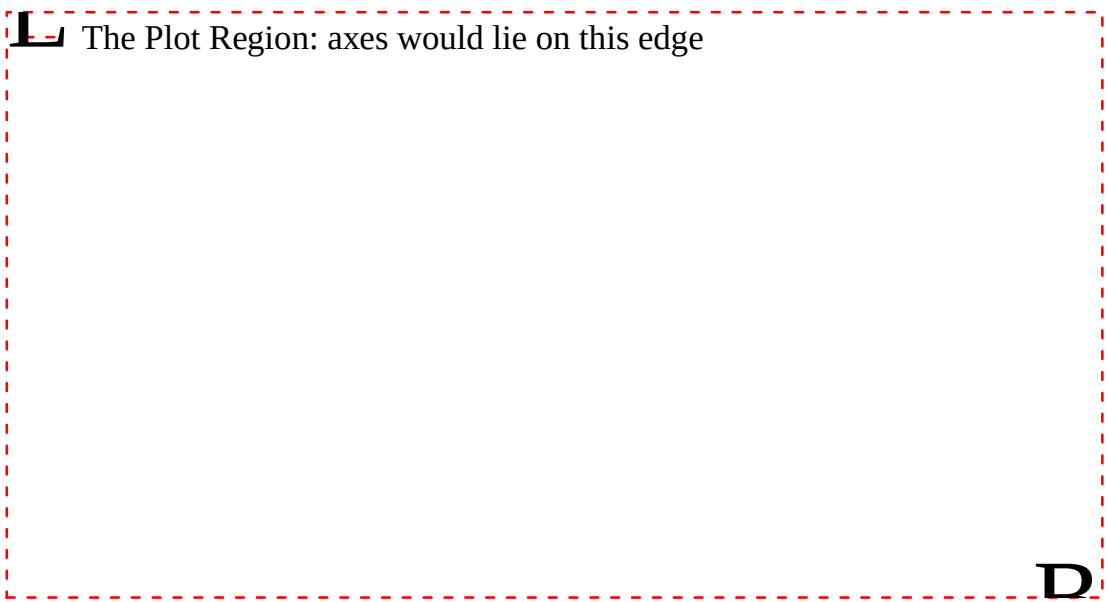
Some monkey business at the outer edges

- Ask the plot for the "plot region's dimensions"

```
(pu <- par("usr"))
```

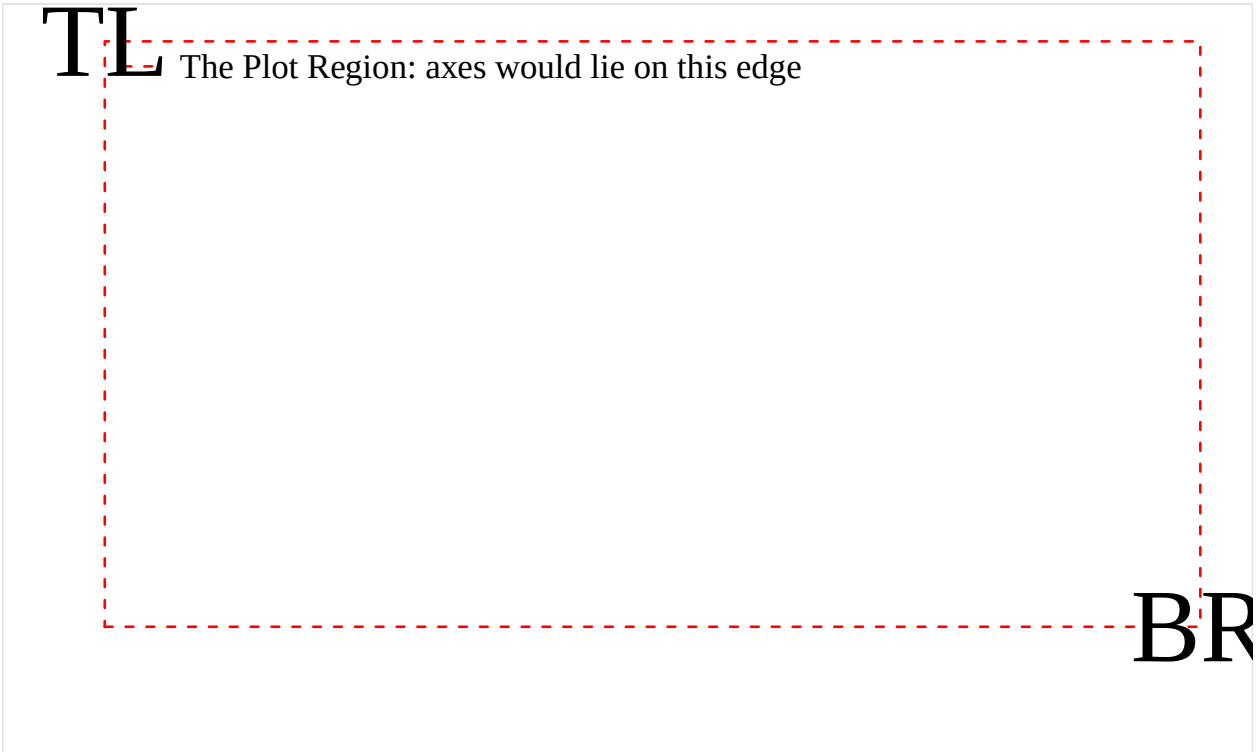
```
[1] 13.72 21.28  4.16 26.84
```

```
text(pu[1], pu[4], label = paste("TL"), cex = 3)
text(pu[2], pu[3], label = paste("BR"), cex = 3)
```



Color outside the Lines

- Run "par(xpd = TRUE)" before the plot commands if you need to write outside the plot region.



Keep decorating

- Put plotting character 18 at the means.

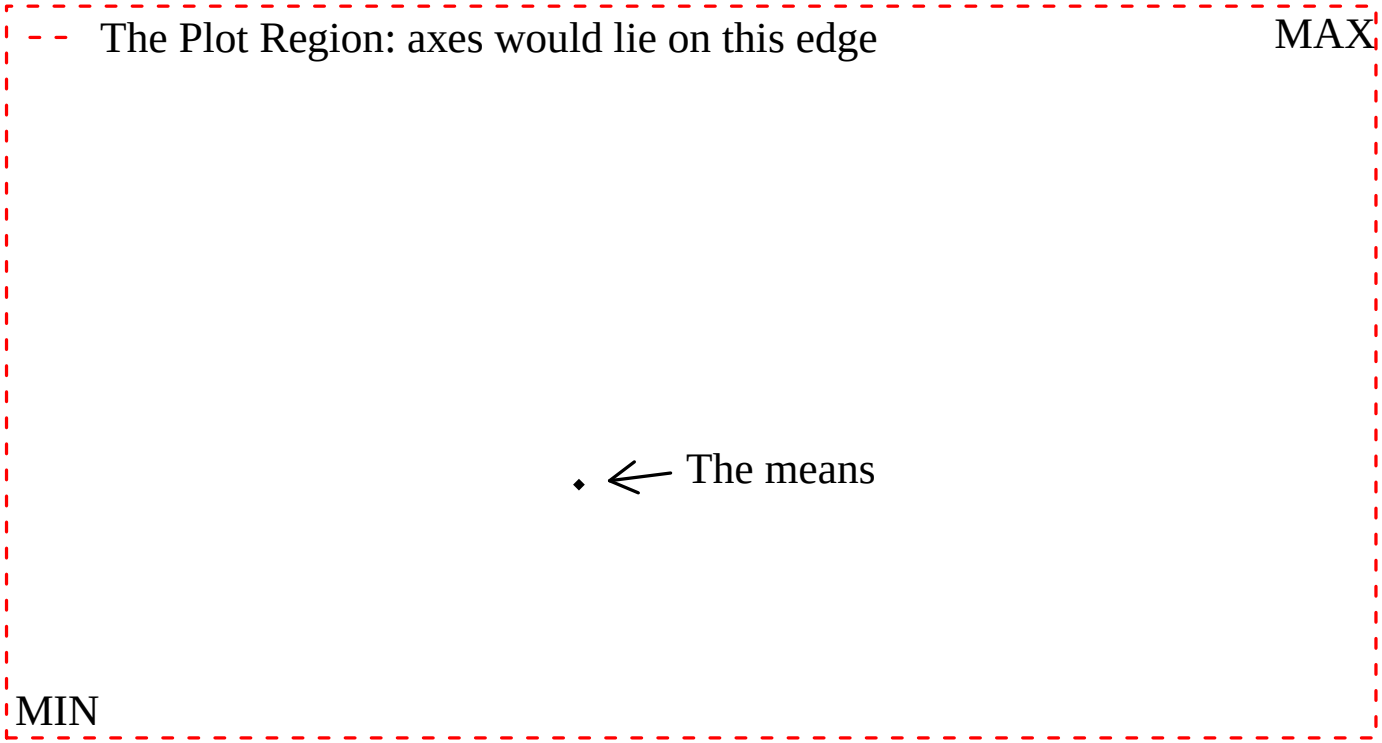
```
ma <- round(mean(mydf$age), 2)
ms <- round(mean(mydf$salary), 2)
points(ma, ms, pch=18)
```

- This will draw an arrow pointing at the means

```
arrows(1.01*ma, 1.01*ms, 1.03*ma, 1.03*ms, code=1, length=0.1)
```

- This will add a label for the arrow

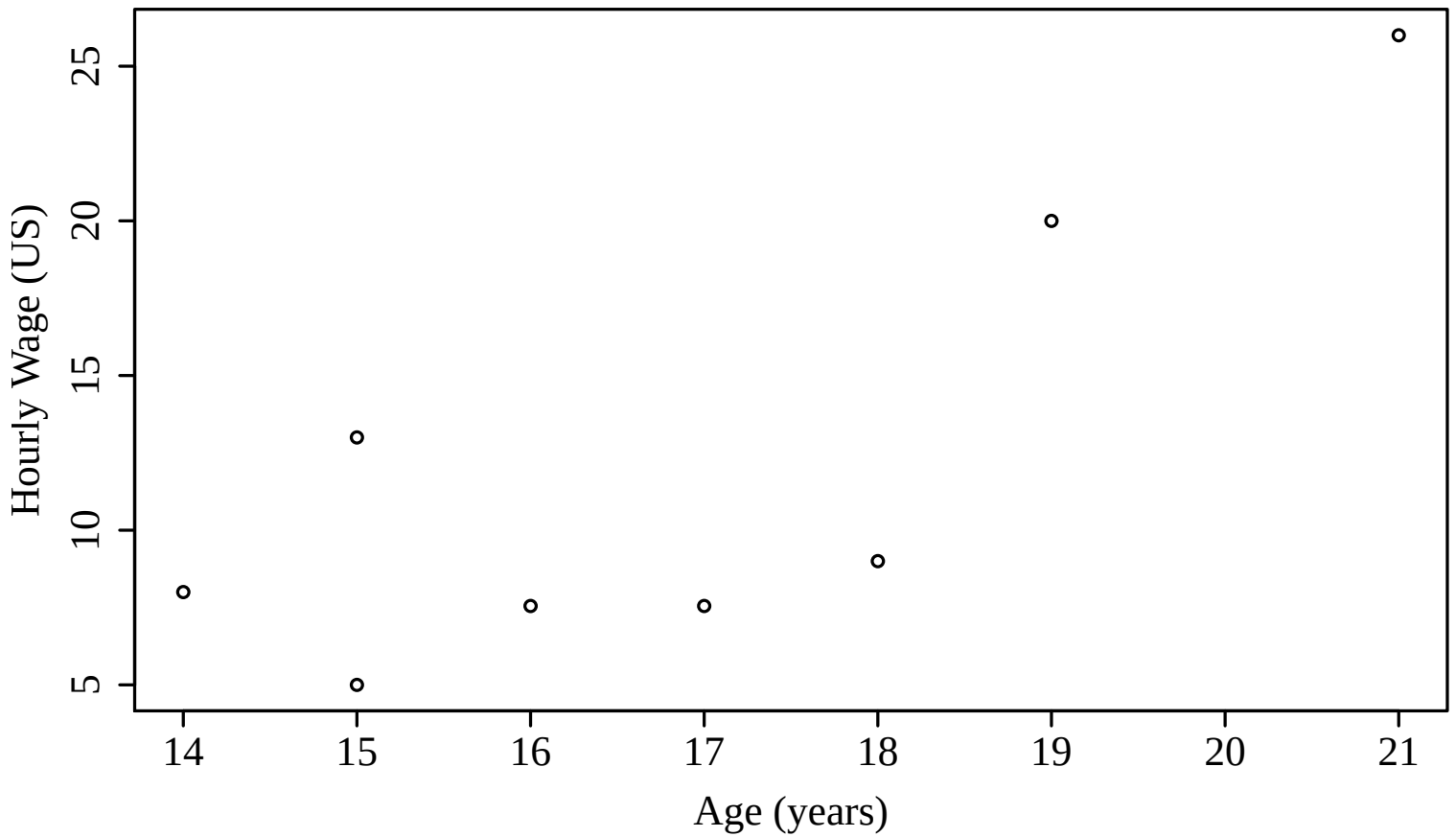
```
text(1.03*ma, 1.03*ms, pos=4, label="The means")
```


The Plot Region: axes would lie on this edge
MAX
The means
MIN

Could we Reproduce the Original Scatter?

```
plot(salary~age , data=mydf, type="n", bty="n", axes=F, xlab="",
     ylab="")
axis(1); mtext("Age (years)", side = 1, line = 3)
axis(2); mtext("Hourly Wage (US)", side = 2, line = 3)
box()
with(mydf, points(age, salary , pch=1, cex=1))
```

Could we Reproduce the Original Scatter?



Outline

Please Try This

- First, generate some example data

```
set.seed(1234321)
myx <- rnorm(1000, mean=50, sd=20)
```

- Then explore myx with these commands

```
rockchalk::summarize(myx) ## or summary(myx)
hist(myx)
dev.new(height=3, width=8) ## fiddle values
hist(myx)
hist(myx, breaks = seq(min(myx), max(myx), length = 5))
hist(myx, breaks = c(min(myx), 0, 40, 50, max(myx)))
hist(myx, xlim = c(-200, 200))
hist(myx, xlim = c(-200, 200), ylim = c(0, 350))
text(-200, 300, pos = 4, labels = c("If you typed all this \nin
     I think you are the \nmost dedicated and well \nadjusted
    person who ever \ntook my course"))
```

Let's Beautify a Histogram!

- The expected value is 50, standard deviation is 20

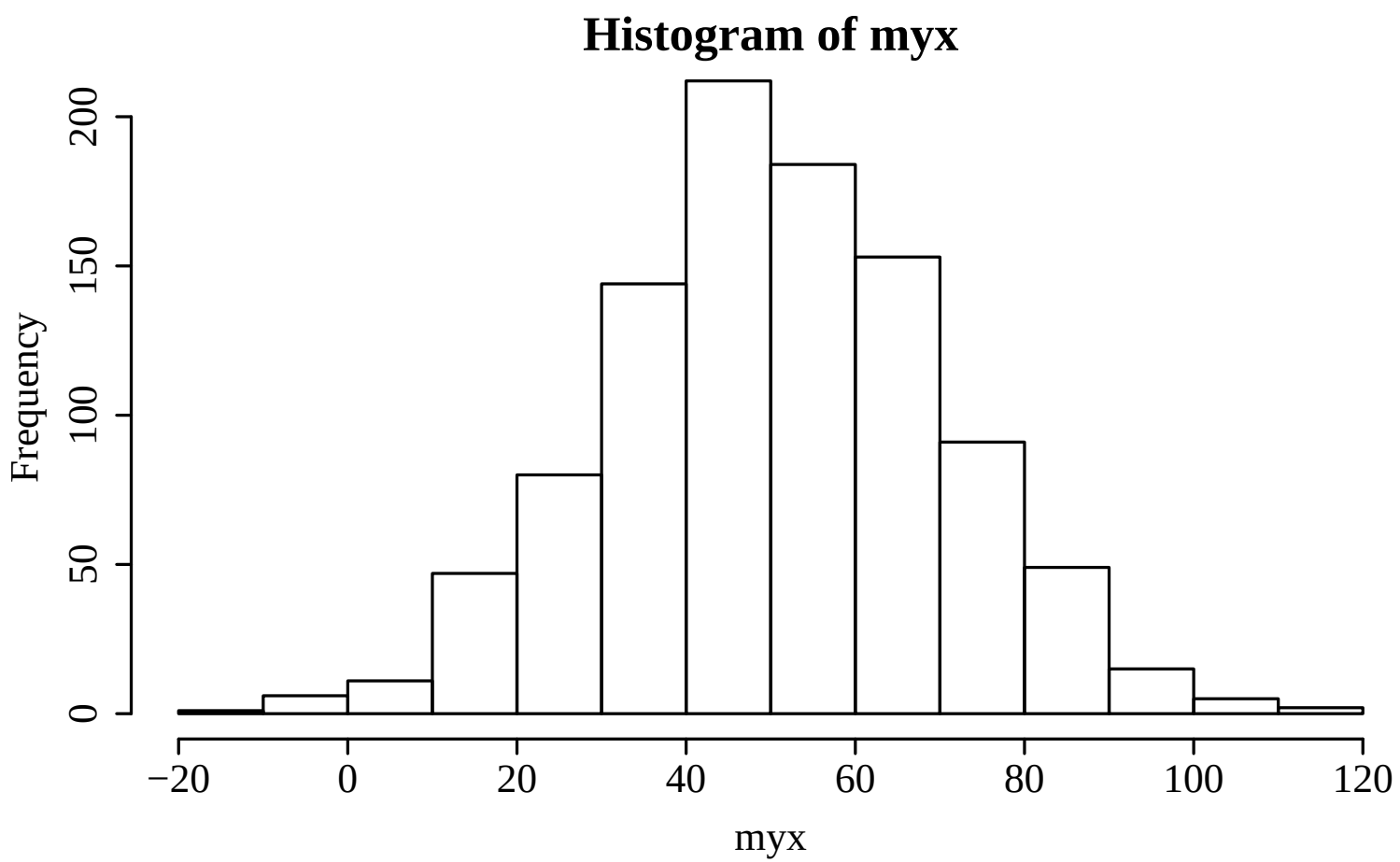
```
set.seed(1234321)
myx <- rnorm(1000, mean=50, sd=20)
```

- Incidentally, observed mean is 50.4845385676878, std. dev.=19.9768673755888
- I'll need the observed mean and standard deviation later:

```
myxm <- mean(myx)
myxsd <- sd(myx)
```

Create A Histogram

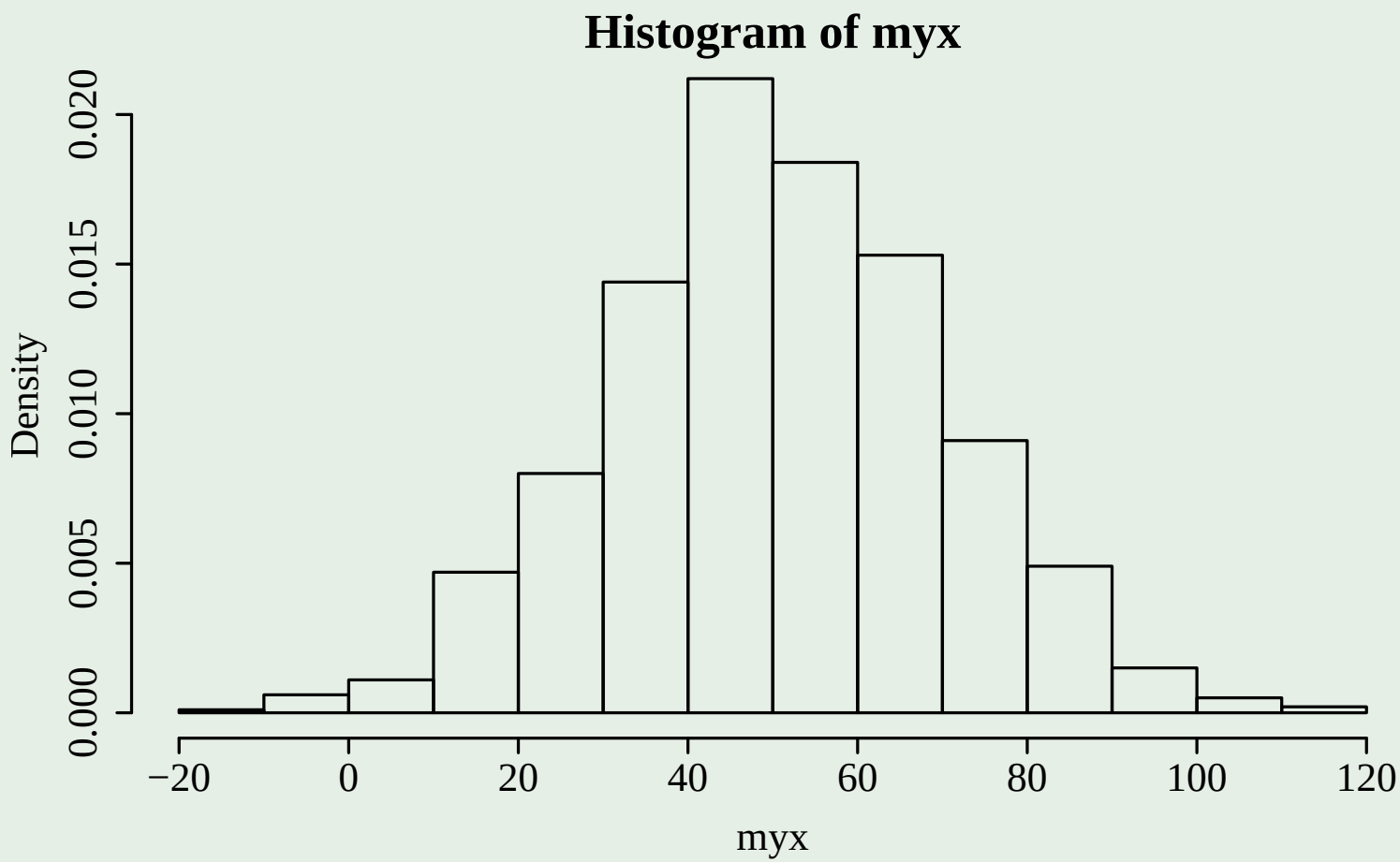
```
hist(myx)
```



That's not awesome yet

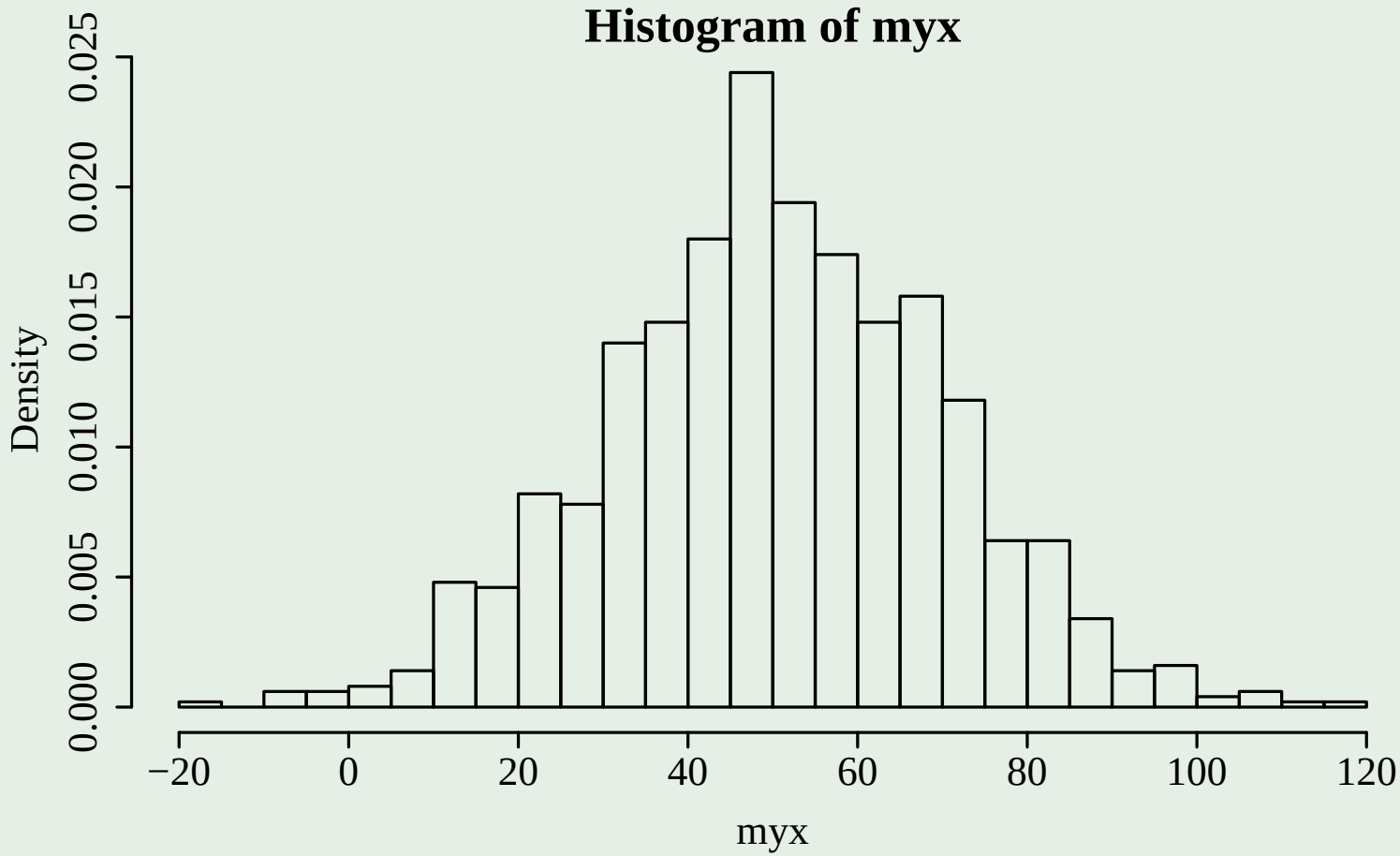
Tell it you want "density" values, not counts

```
hist(myx , prob=T)
```



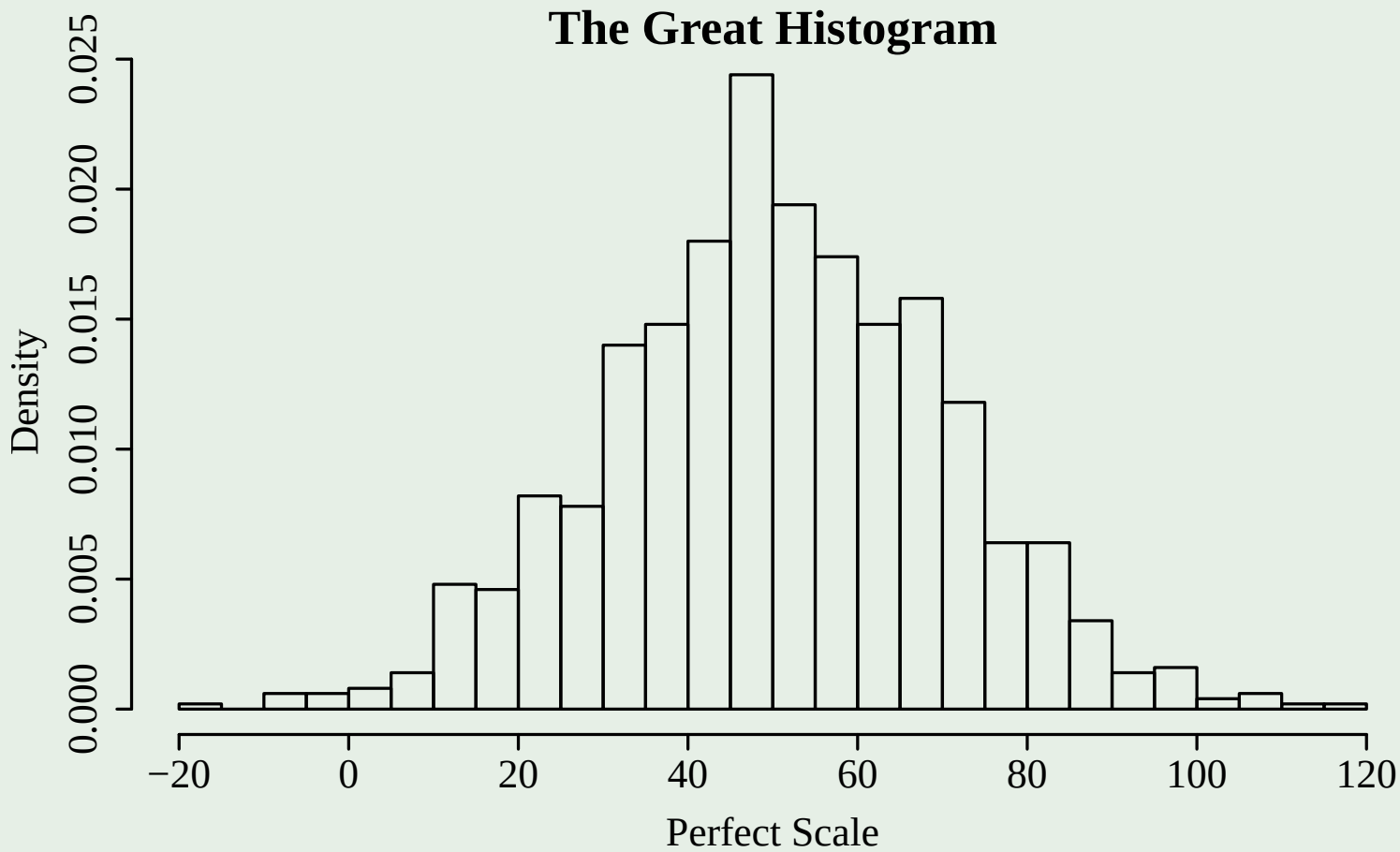
I like narrower bars, so specify breaks

```
hist(myx , breaks=30, prob=T)
```



We Might As Well Be Informative

```
hist(myx , breaks=30, prob=T, xlab="Perfect Scale", ylab="Density",
     main="The Great Histogram")
```



How Do I Know Hist Can Do Those Things?

- Experience: Most R plot methods have same/similar options
- Read ?hist
- try this: RSiteSearch("histogram")

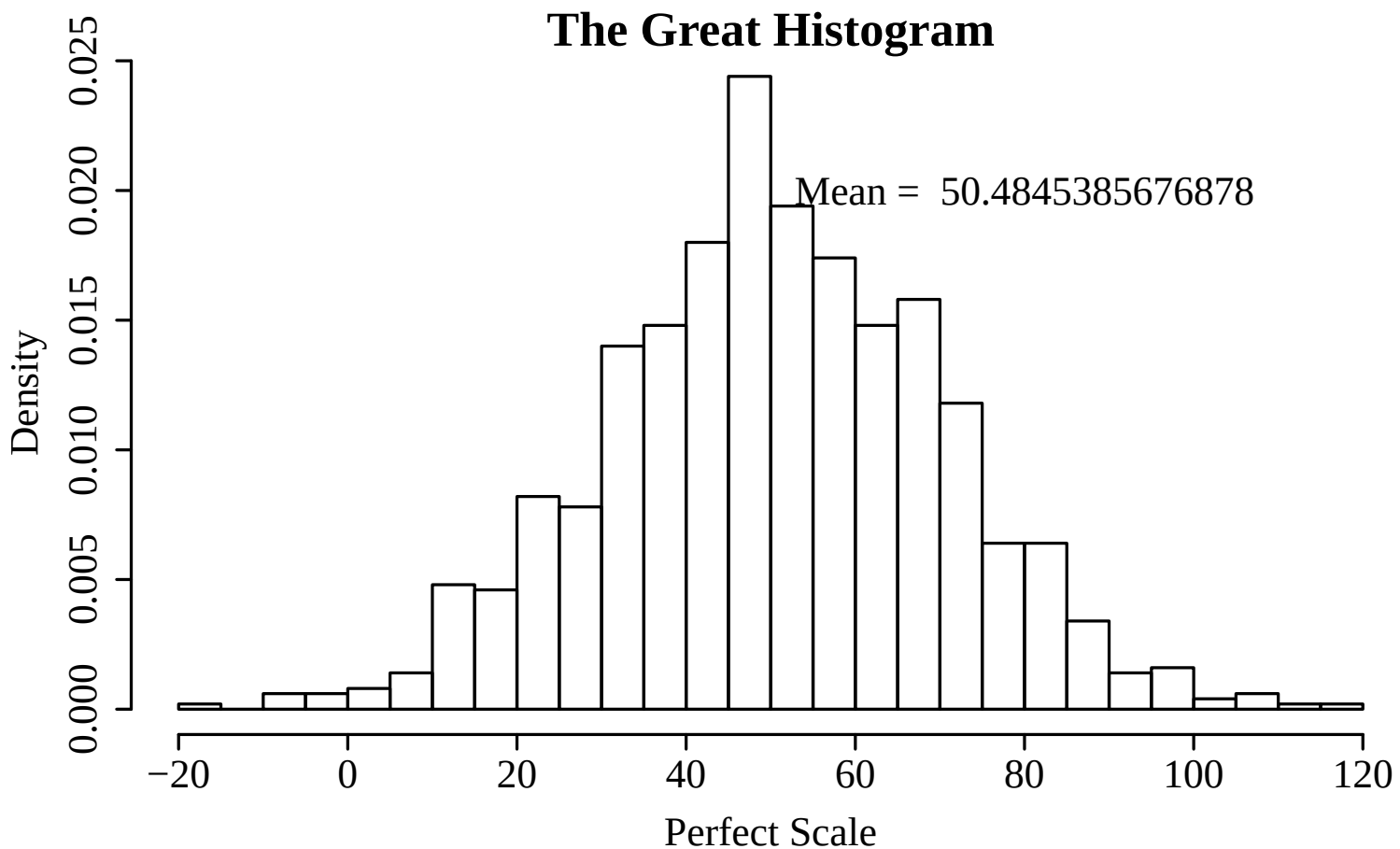
Now Add Some Annotation

- Re-run the hist command to re-draw the whole graph
- Use the text() command to write the mean in the "white space"
- text command has this format
  > text(x, y, label="some word")
- In this case, I used

```
text(80, 0.02, label=paste("Mean = ", myxm))
```

- Note text is "vectorized" so you can give it vectors for x,y, and label and it will write all of those combinations.

Oh, Bother. Too Many Small Numbers



Maybe we like to see the standard deviation as well?

- I am tempted to round the variables ahead of time,

```
myxm <- round(myxm, 2)
myxsd <- round(myxsd, 2)
```

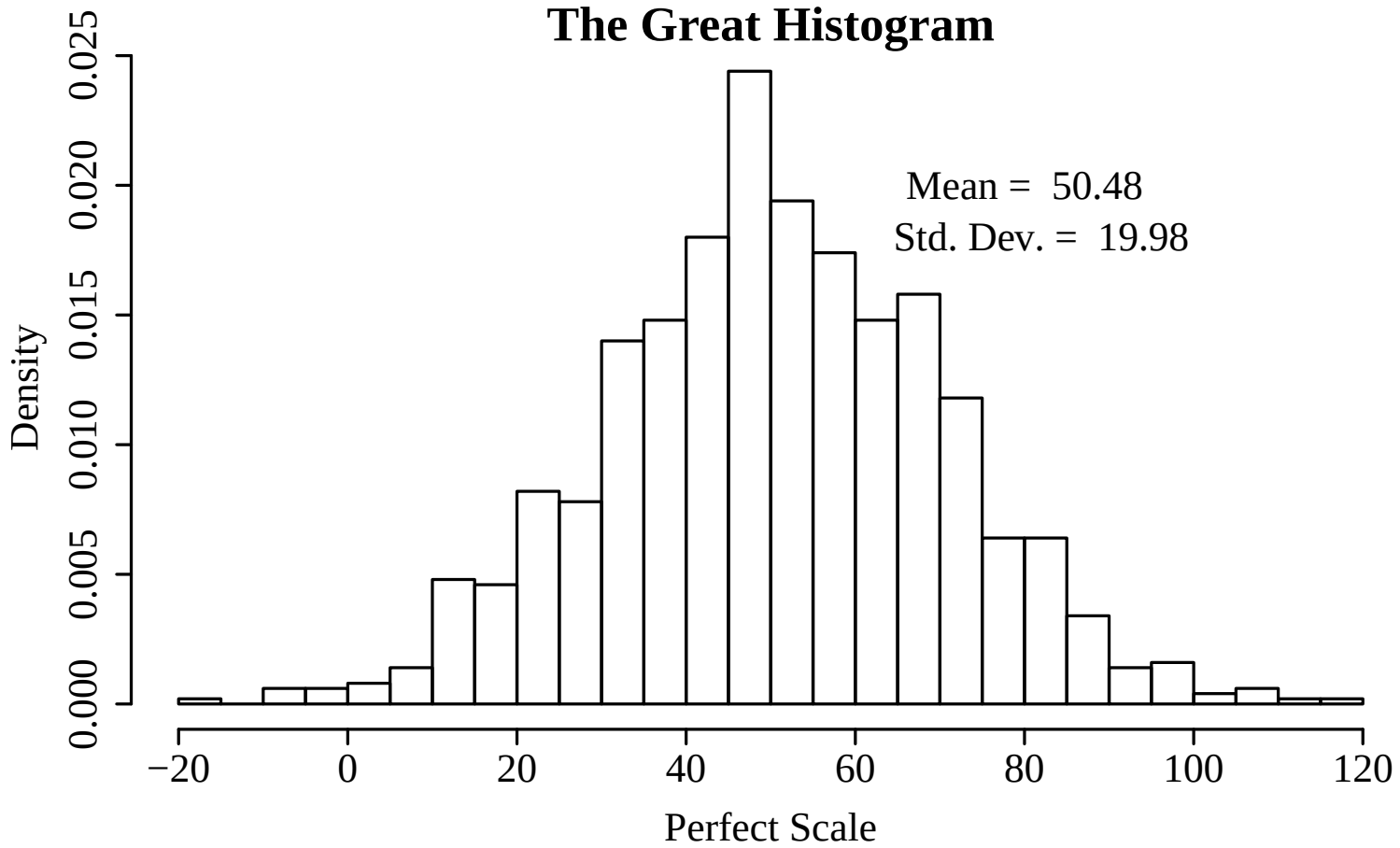
  but I'm reluctant, so I'll round the numbers when I need them.

- First, the figure has to be redrawn

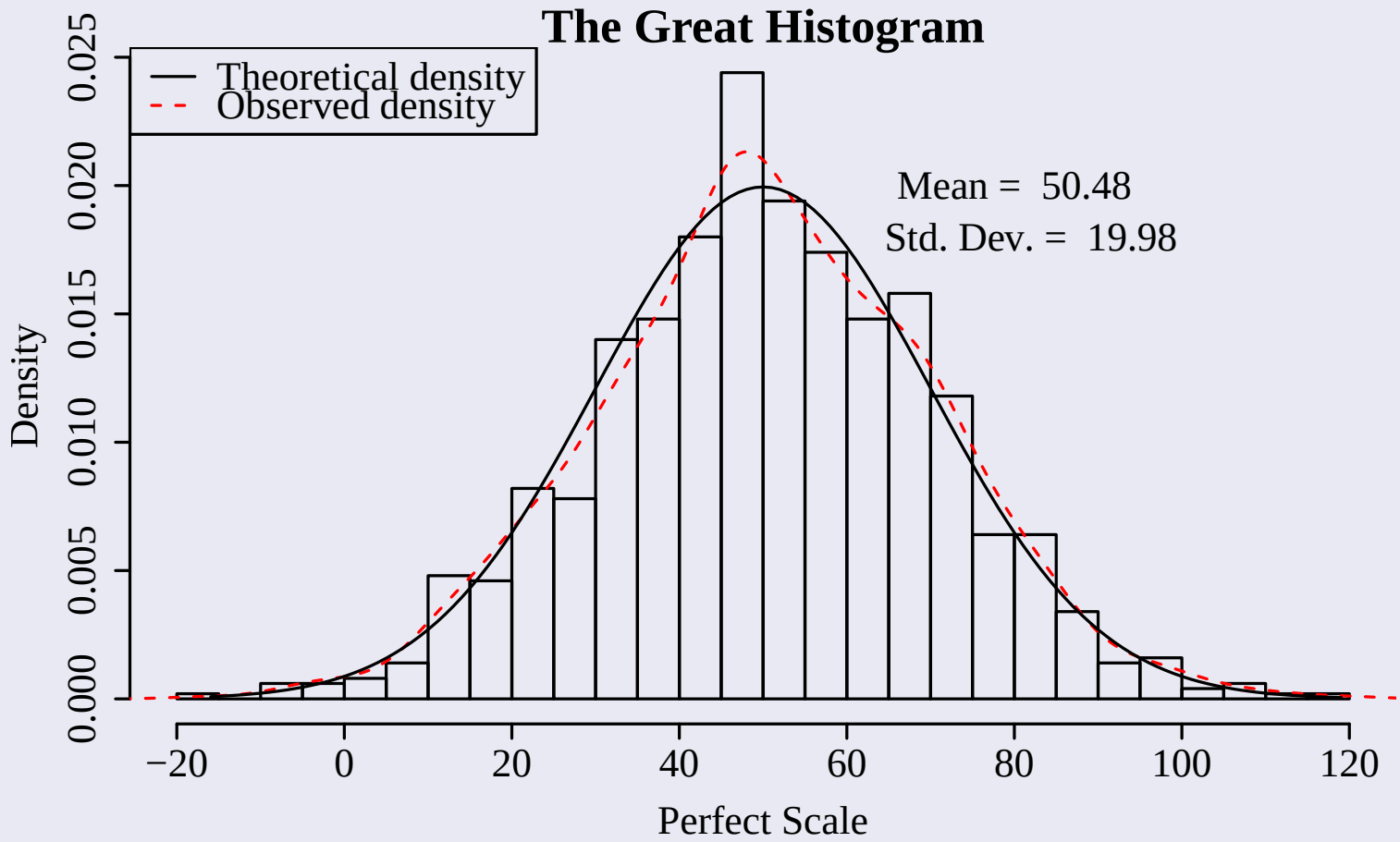
```
hist(myx , breaks=30, prob=T, xlab="Perfect Scale", ylab="
    Density", main="The Great Histogram")
```

- Then add the corrected text commands

```
text(80, 0.020, label=paste("Mean = ", round(myxm,2)))
text(82, 0.018, label=paste("Std. Dev. = ", round(myxsd,2)))
```


The Great Histogram
Mean = 50.48
Std. Dev. = 19.98
Density
0.000
0.005
0.010
0.015
0.020
0.025
−20
0
20
40
60
80
100
120
Perfect Scale

As Good as it Gets: Add Density Lines and a Legend



Wow. How did he do that density line?

```
denx <- density(myx)
lines(denx, lty=2, col="red")
```

Explanation:

- Use `density()` to do whatever it does :)
- Use the `lines()` to do whatever it does :)
  - lty is "line type"
  - col is "color"
- Somehow `lines()` notices we gave it a a density object and "knows what to do".

What about the pdf line?

This is tougher. lines is going to want this information
lines(xcoordinates, ycoordinates)
so it can "connect the dots". And we don't have it.

```
rangx <- range(myx)
testseq <- seq(rangx[1], rangx[2], by=1)
pdfseq <- dnorm(testseq, mean=50, sd=20)
lines(testseq, pdfseq)
```

- Create xcoordinates by getting the range of the observed x
- and then create a sequence between the min and max
- Give that sequence to the "dnorm" function, which calculates the "true density value".
- Supply x, y to lines()

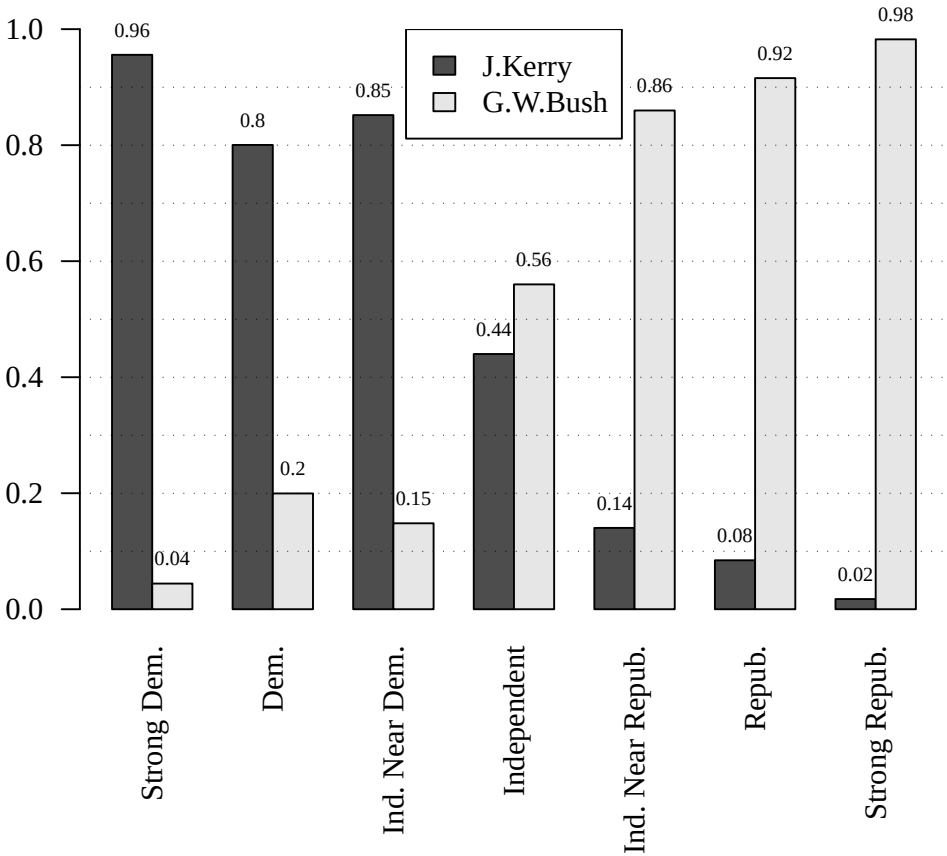
And The Legend. Its so Awesome!

- Ordinarily, legend wants to know coordinates, x and y
- Shortcuts have been created, like “topleft” or “topright”, to replace x,y.
- Everything else is detail work

```
legend("topleft", legend = c("Theoretical density", "Observed
    density"), lty = c(1,2), col = c("black","red"))
```

Outline

Perhaps the Finest Barplot Ever



Before I tell you how to do that...

I Want You To Do This:
In the folder
`http://pj.freefaculty.org/R/WorkingExamples`
you should find lot of R files. Find the one called
`plot-barplot-1.R`
Open that and step through it.

barplot() wants an R table as input

- Create a table object called t1

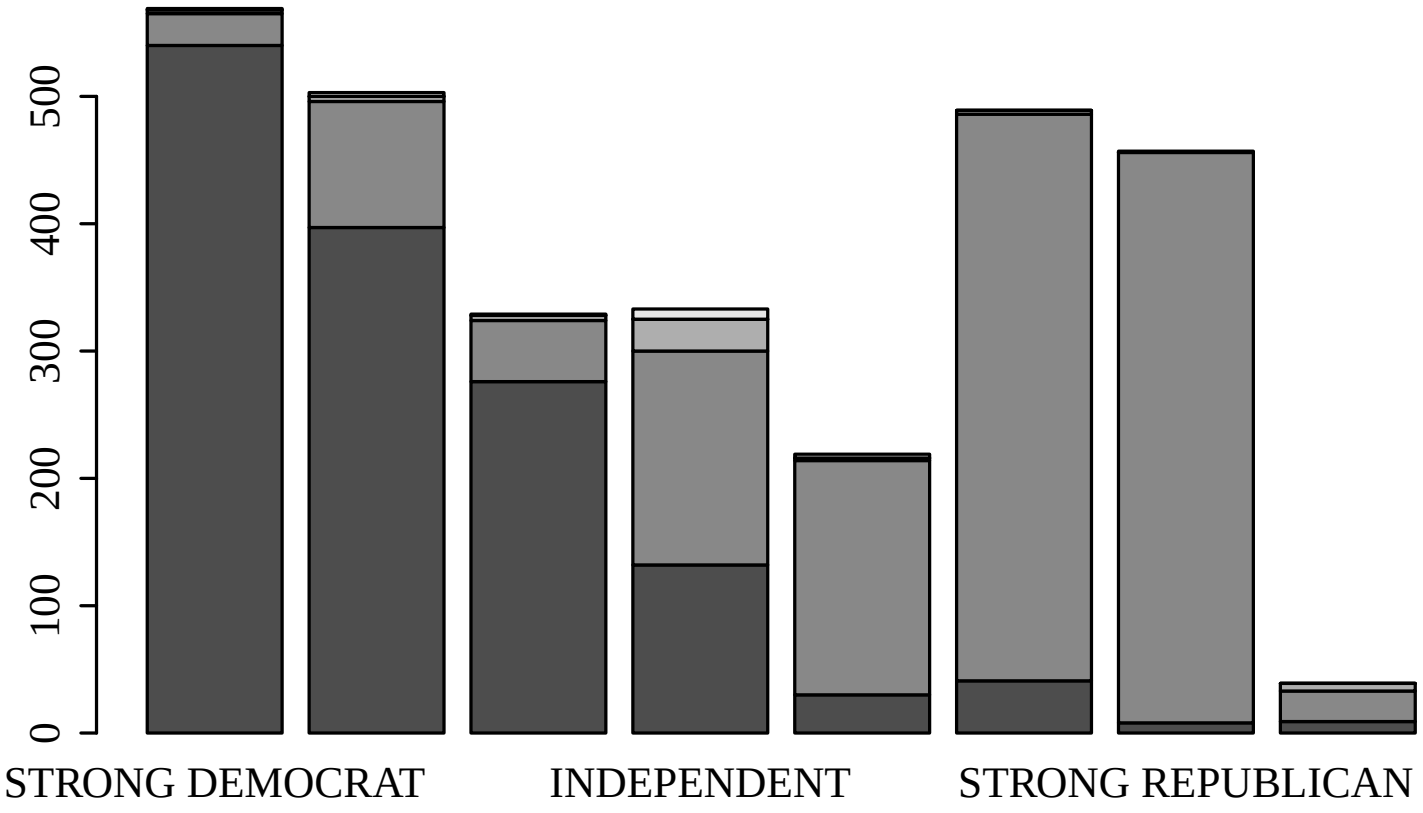
```
t1 <- with(dat, table(pres04,
    partyid))
t1
```

```
                    partyid
pres04               STRONG DEMOCRAT
    NOT STR DEMOCRAT IND,NEAR
    DEM INDEPENDENT IND,NEAR REP
    NOT STR REPUBLICAN STRONG
    REPUBLICAN OTHER PARTY
 KERRY                                540
                        397
                  276           132
                   30
                         41
                         8
                   9
 BUSH                                  25
                        99
                   48           168
                  184
                        445
                        448
```

```
                  24
NADER                                  3
                       4
                   4              25
                   2
                     3
                     0
                  6
OTHER (SPECIFY)                        0
                     0
                   0                 0
                   0
                     0
                     0
                  0
DIDNT VOTE                             1
                     3
                   1                 8
                   3
                     0
                     1
                  0
```

But that makes an ugly barplot

```
barplot(t1)
```



First, clean up partyid, a factor variable

```
levels(dat$partyid)
```

```
[1] "STRONG DEMOCRAT"     "NOT STR DEMOCRAT"    "IND,NEAR DEM"          "
    INDEPENDENT"          "IND,NEAR REP"        "NOT STR REPUBLICAN"  "
    STRONG REPUBLICAN"    "OTHER PARTY"
```

Set other to missing, beautify text for levels

```
plev <- levels(dat$partyid)
dat$partyid[dat$partyid %in% plev[8]] <- NA
dat$partyid <- factor(dat$partyid)
levels(dat$partyid) <- c("Strong Dem.", "Dem.", "Ind. Near Dem.", "
    Independent", "Ind. Near Repub.", "Repub.", "Strong Repub.")
```

Second, clean up the pres04 variable

```
levels(dat$pres04)
```

```
[1] "KERRY"                "BUSH"                 "NADER"
    "OTHER (SPECIFY)"      "DIDNT VOTE"
```

Get rid of minor candidates

```
preslev <- levels(dat$pres04)
dat$pres04[dat$pres04 %in% preslev[3:10]]<- NA
dat$pres04 <- factor(dat$pres04)
levels(dat$pres04) <- c("Kerry", "Bush")
```

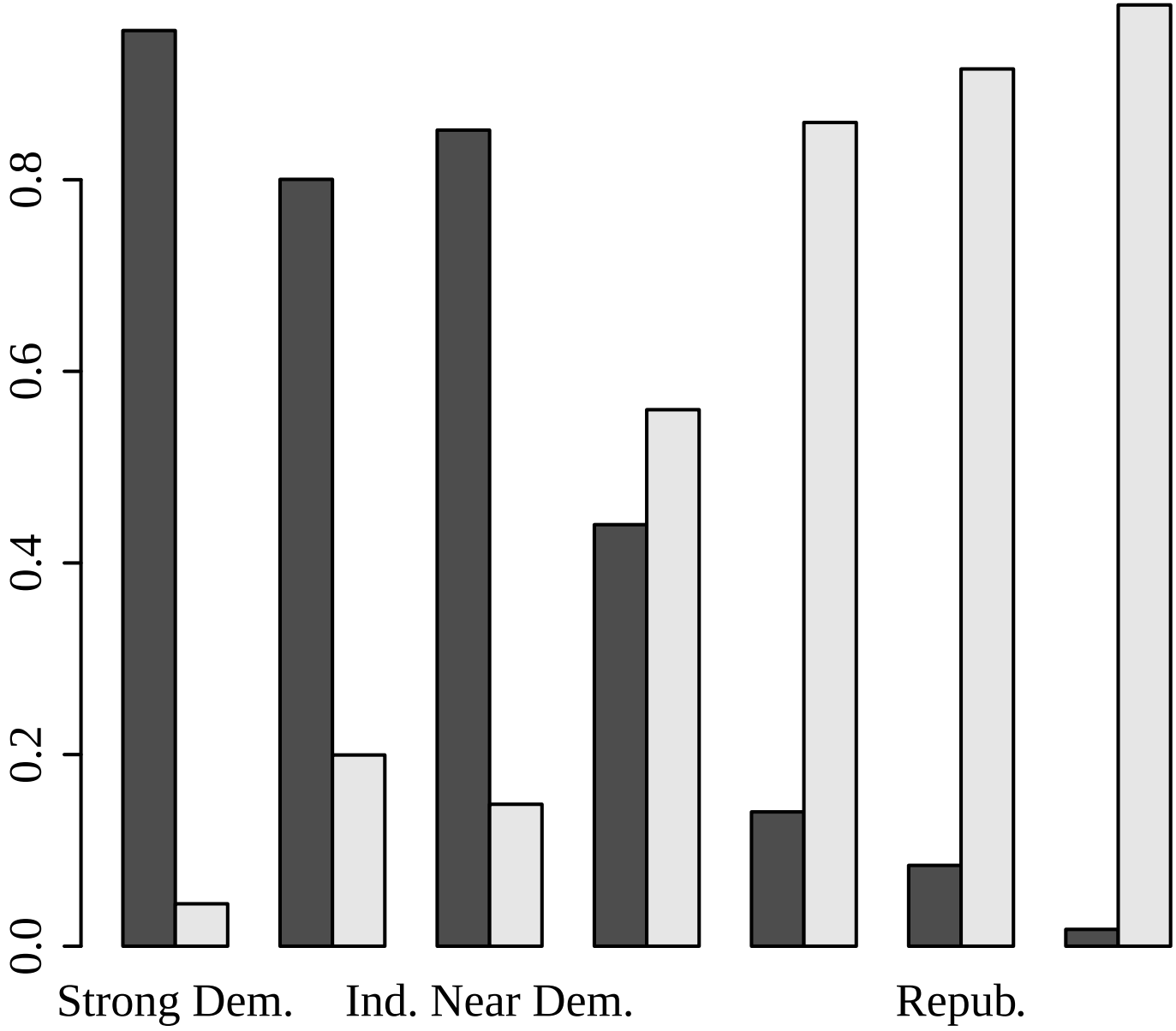
Third, make a new table, with proportions

```
oldtable <- with(dat, table(pres04, partyid))
t1 <- prop.table(oldtable,2)
t1
```

```
        partyid
pres04   Strong Dem.       Dem. Ind. Near Dem. Independent Ind. Near
    Repub.      Repub. Strong Repub.
  Kerry  0.95575221 0.80040323     0.85185185  0.44000000        0
       .14018692 0.08436214    0.01754386
  Bush   0.04424779 0.19959677     0.14814815  0.56000000        0
       .85981308 0.91563786    0.98245614
```

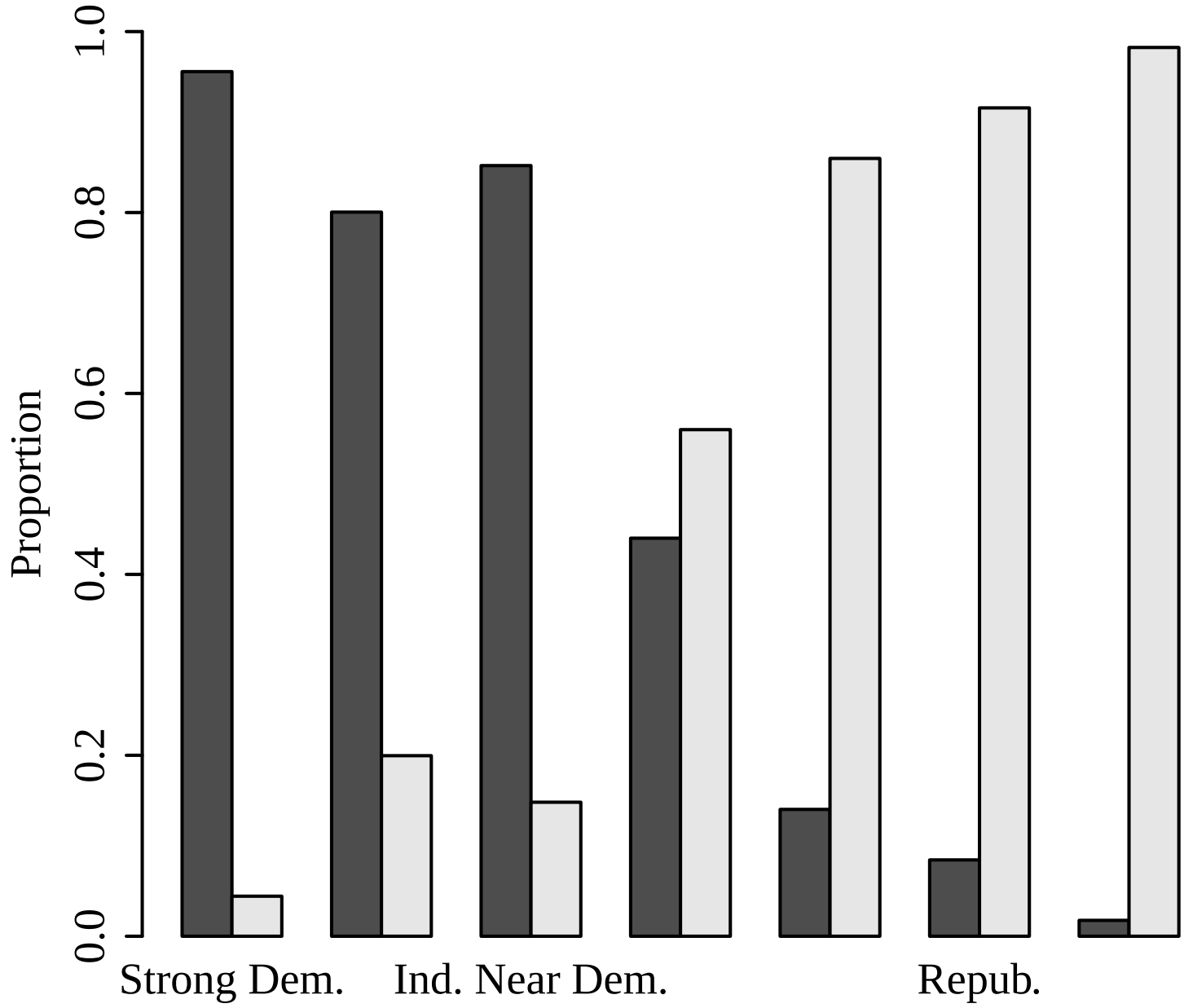
I prefer "beside" barplots

```
barplot(t1, beside=T)
```


0.0
0.2
0.4
0.6
0.8
Strong Dem.
Ind.
Near Dem.
Repub.

I want the y axis to have range from 0 to 1

```
barplot(t1, beside=T, ylim=c(0,1), ylab="Proportion")
```


Proportion
0.0
0.2
0.4
0.6
0.8
1.0
Strong Dem.
Ind.
Near Dem.
Repub.

Turn the labels on X to “vertical” with las=2

```
barplot(t1, beside=T, las=2, ylim=c(0,1), ylab="Proportion")
```

Poop. The X labels run off the bottom

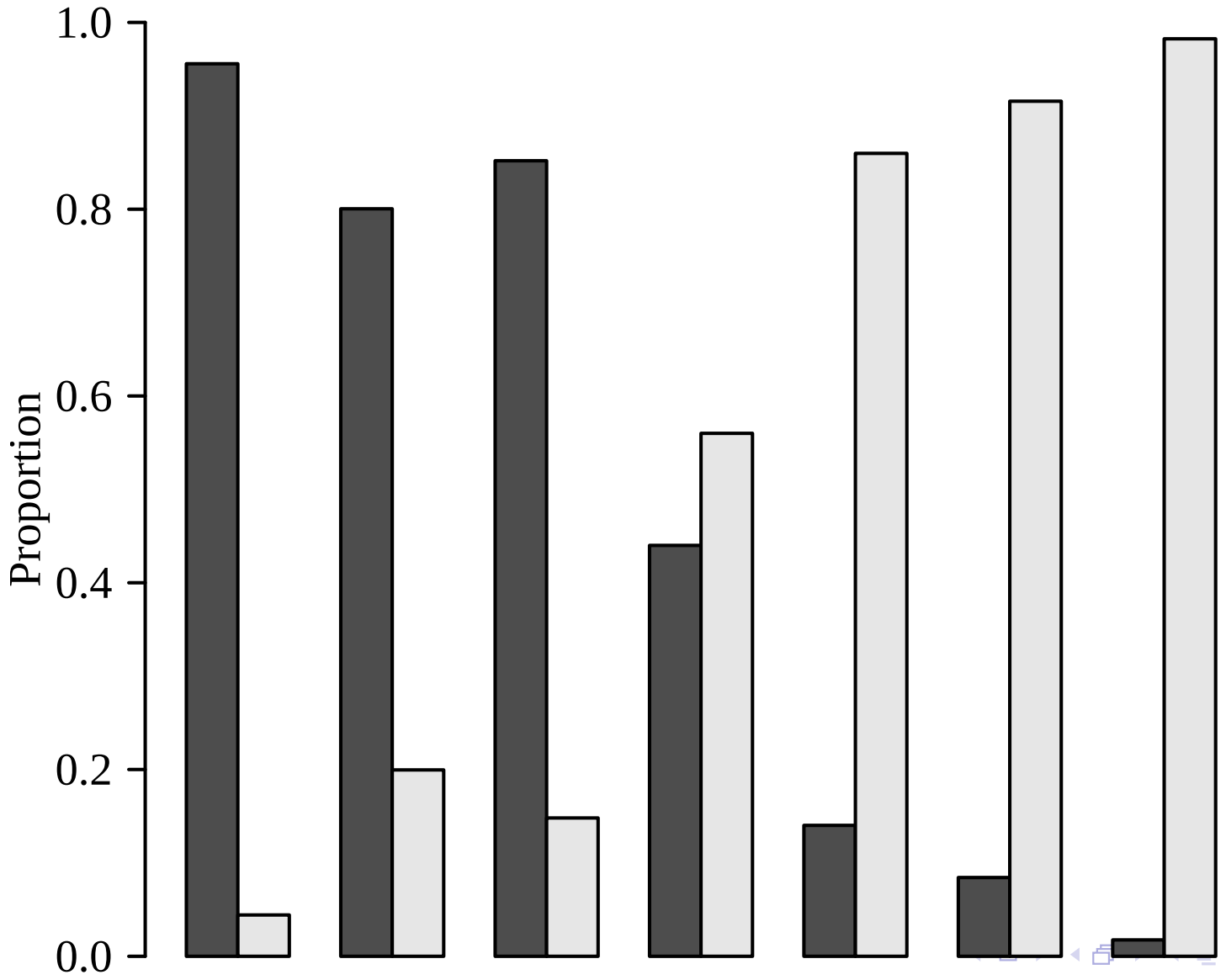
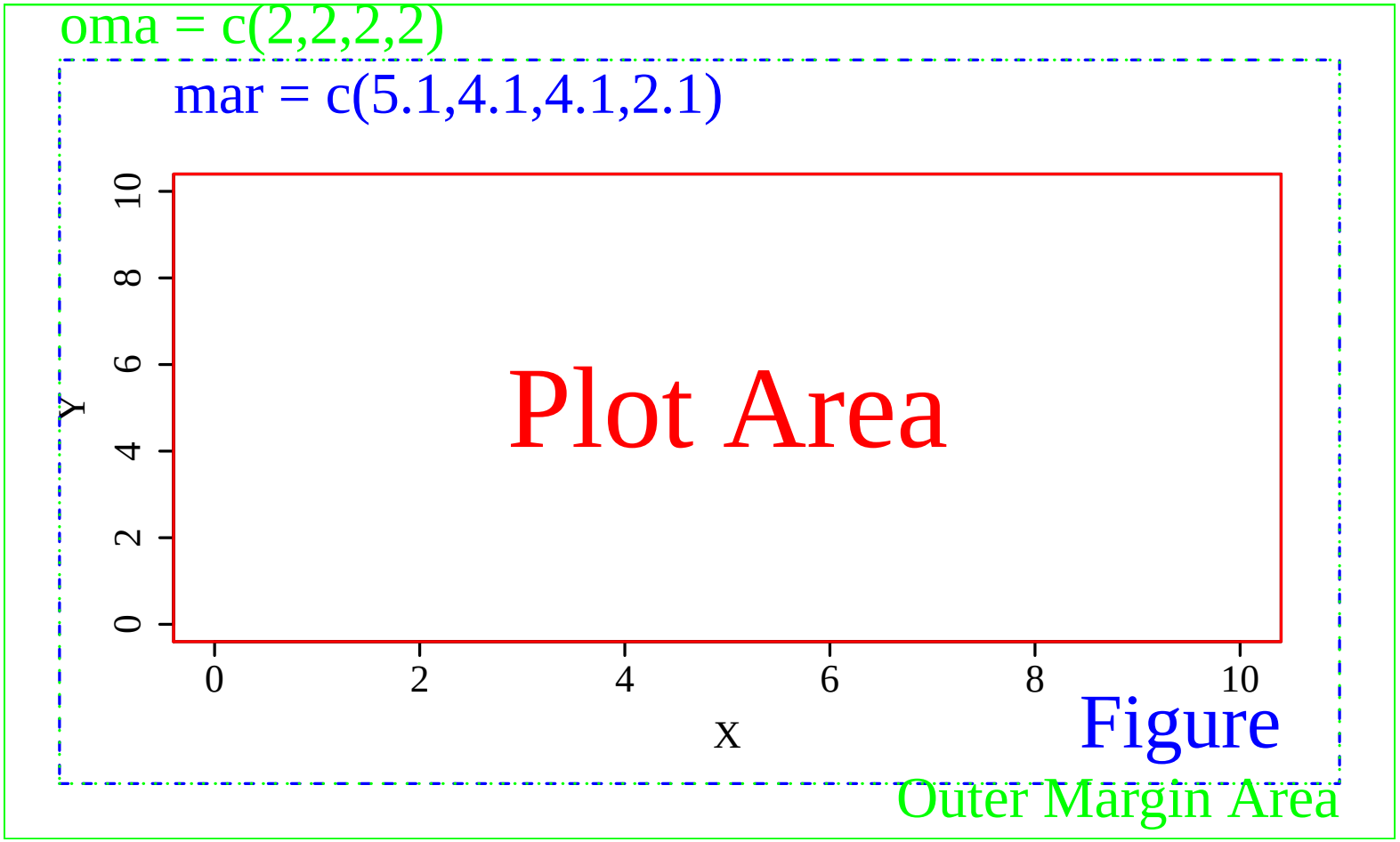
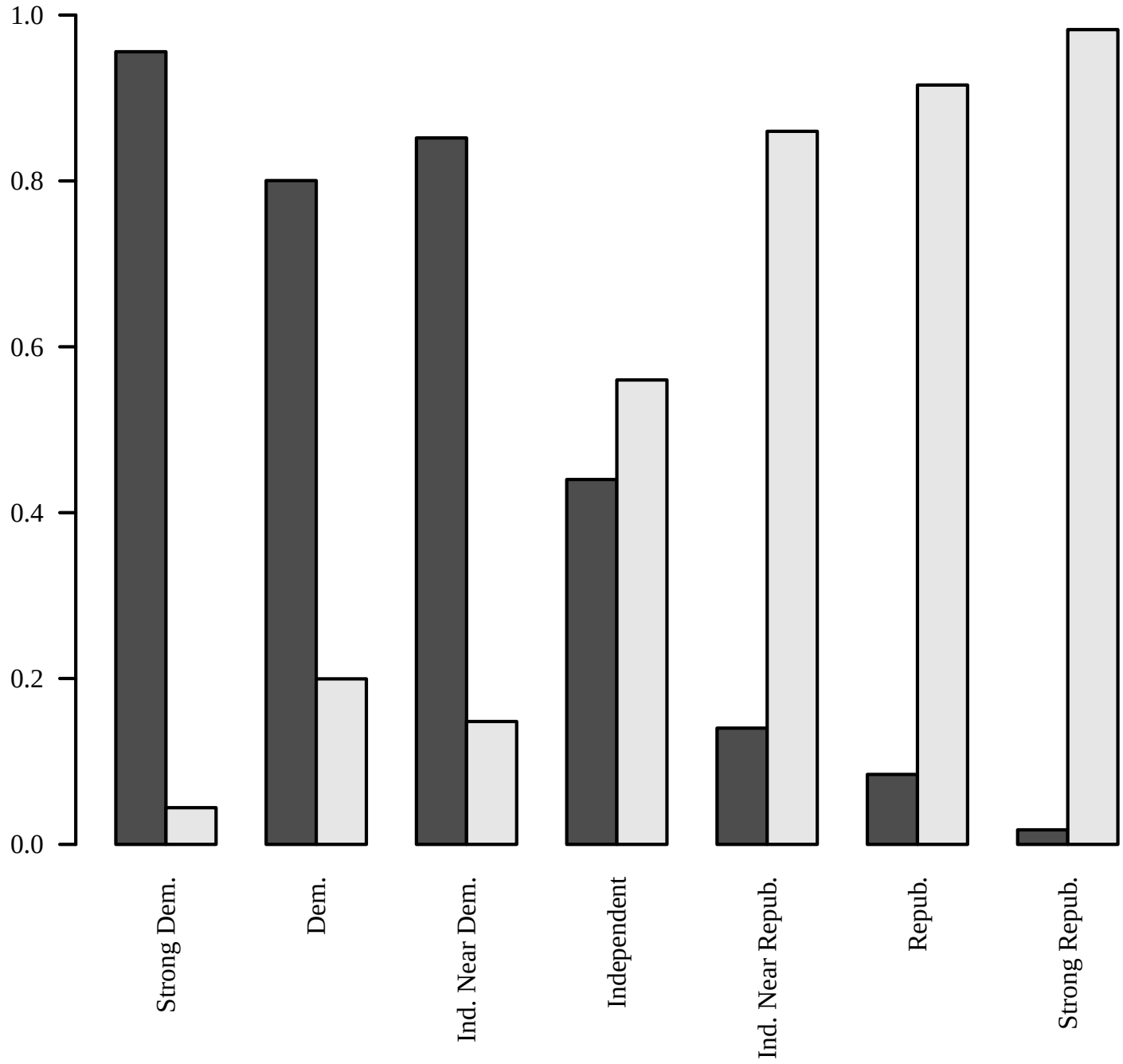


Illustration of Margins in R plot Devices



Use par to make a bigger bottom margin

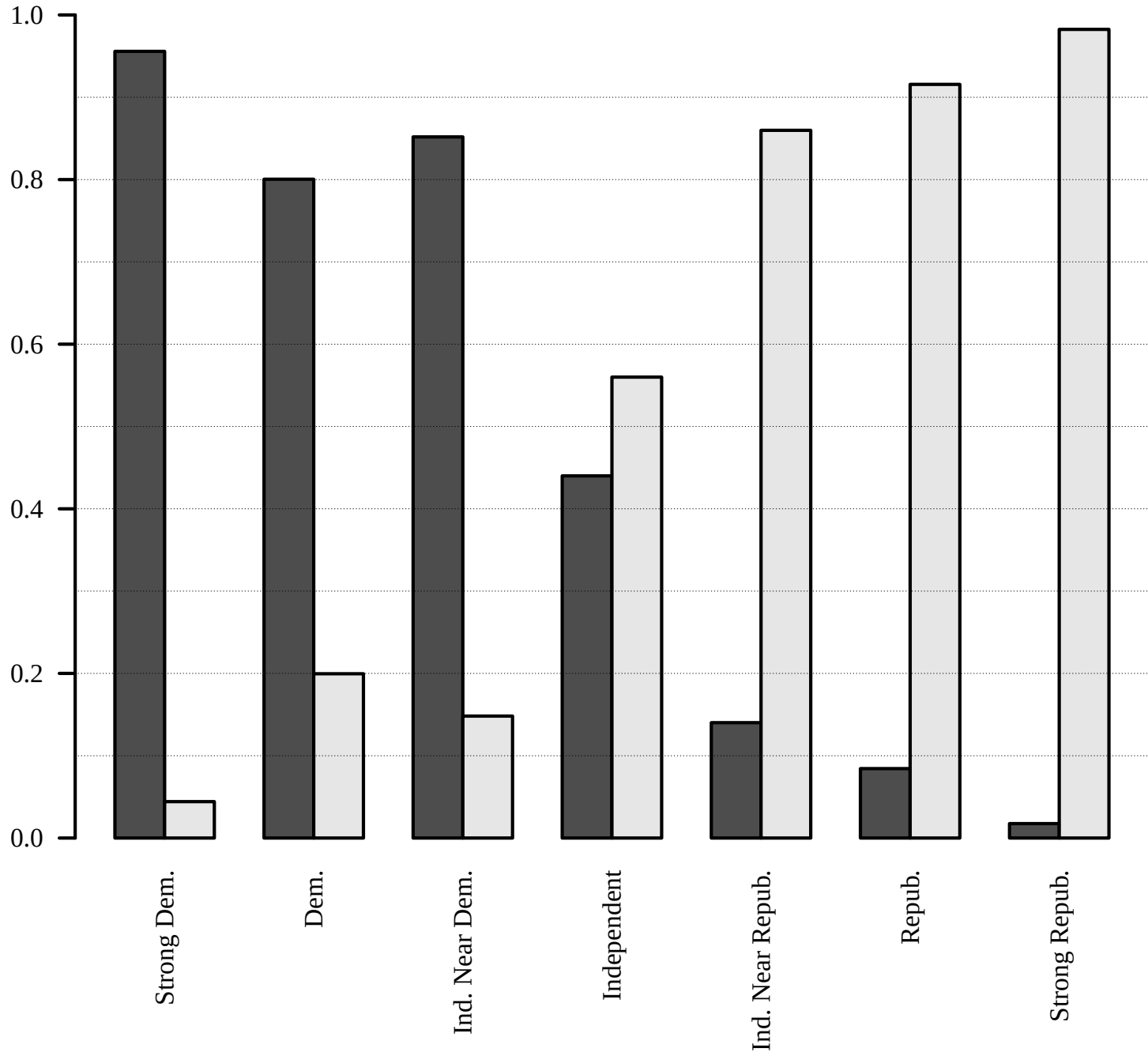
```
par(mar=c(7.1, 4.1, 2, 2.1))
barplot(t1,beside=T, las=2, ylim=c(0,1))
```


1.0
0.8
0.6
0.4
0.2
0.0
Strong Dem.
Dem.
Ind. Near Dem.
Independent
Ind. Near Repub.
Repub.
Strong Repub.

I like those light dotted horizontal lines

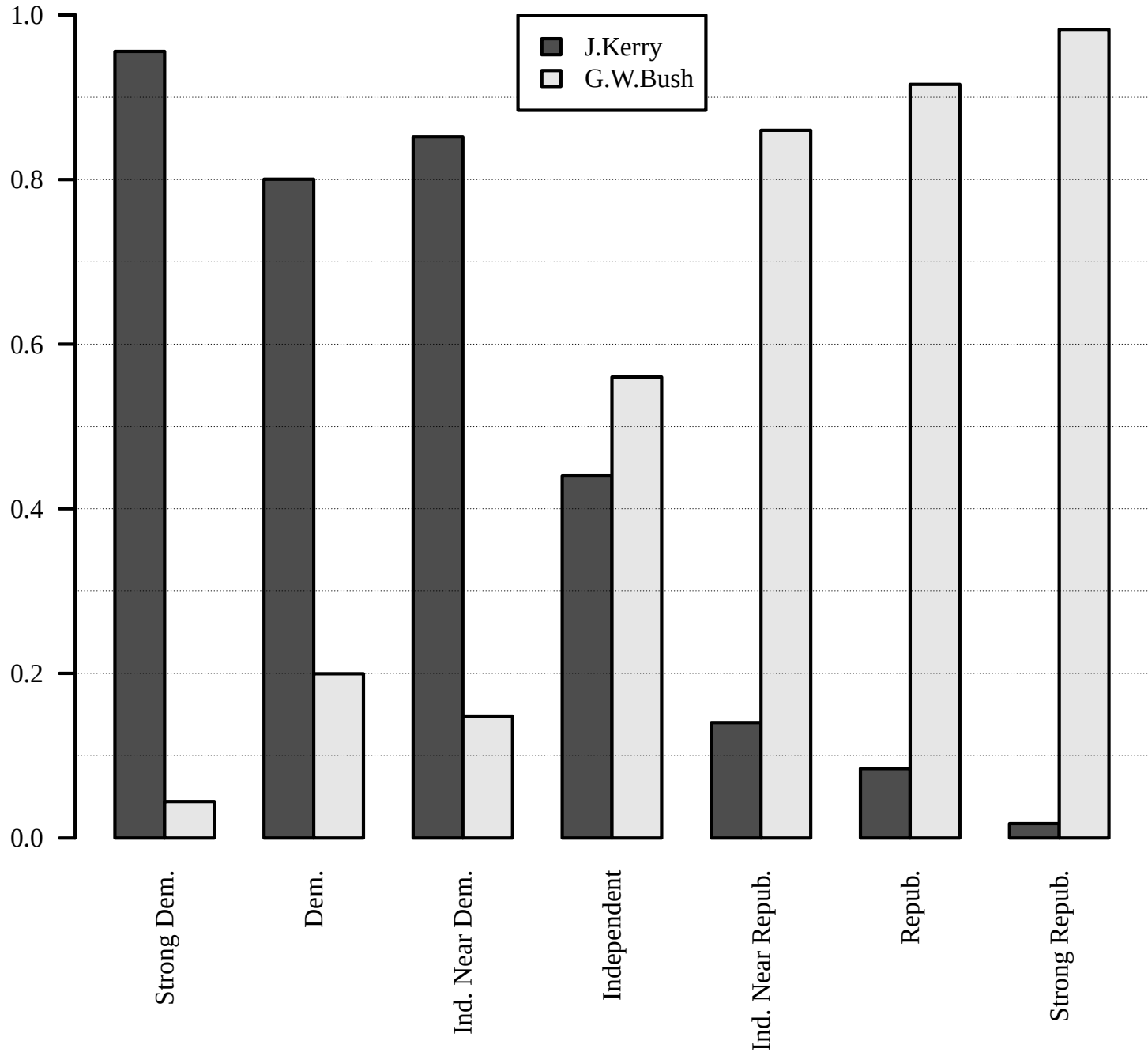
Add this:

```
abline(h=seq(0.1, 0.9, by=0.10), lty=3,  lwd=0.2)
```


1.0
0.8
0.6
0.4
0.2
0.0
Strong Dem.
Dem.
Ind. Near Dem.
Independent
Ind. Near Repub.
Repub.
Strong Repub.

Nobody Hates Legends. Right?

```
legend("top", legend=c("J.Kerry","G.W.Bush"), fill=gray.colors(2),
       bg="white")
```


1.0
0.8
0.6
0.4
0.2
0.0
J.Kerry
G.W.Bush
Strong Dem.
Dem.
Ind. Near Dem.
Independent
Ind. Near Repub.
Repub.
Strong Repub.

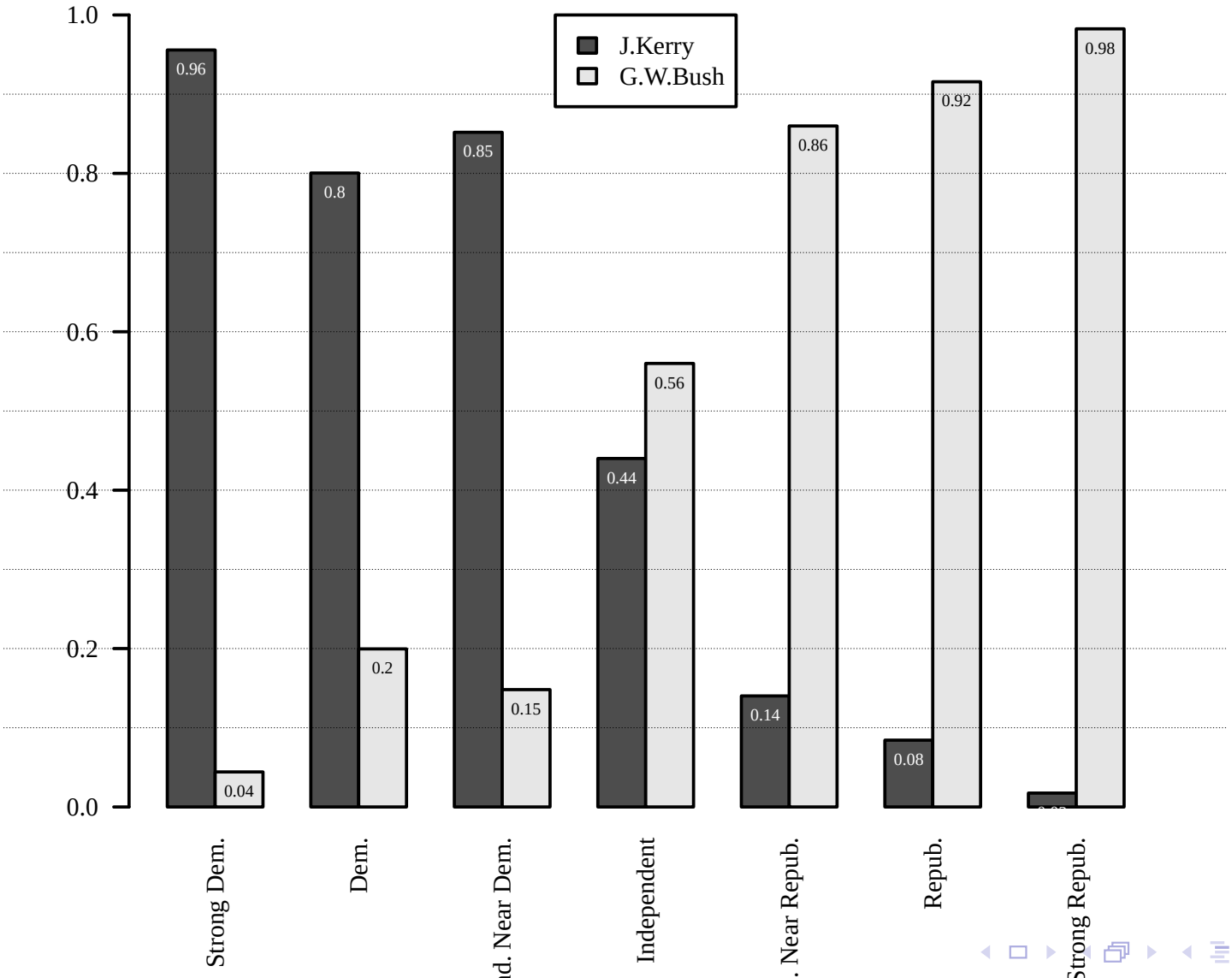
Write inside the bars

```
par(mar=c(7.1, 4.1, 2, 2.1))
par(xpd=T) ##write outside plot region
bp1 <- barplot(t1,beside=T, las=2, ylim=c(0,1))
abline(h=seq(0.1, 0.9, by=0.10), lty=3, lwd=0.2)
legend("top", legend=c("J.Kerry","G.W.Bush"), fill=gray.colors(2),
    bg="white")
bp1
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]  1.5  4.5  7.5 10.5 13.5 16.5 19.5
[2,]  2.5  5.5  8.5 11.5 14.5 17.5 20.5
```

```
text(bp1,t1,label=round(t1,2), pos=1,cex=0.7,col=rbind(rep("white"
    ,7),rep("black",7)))
```

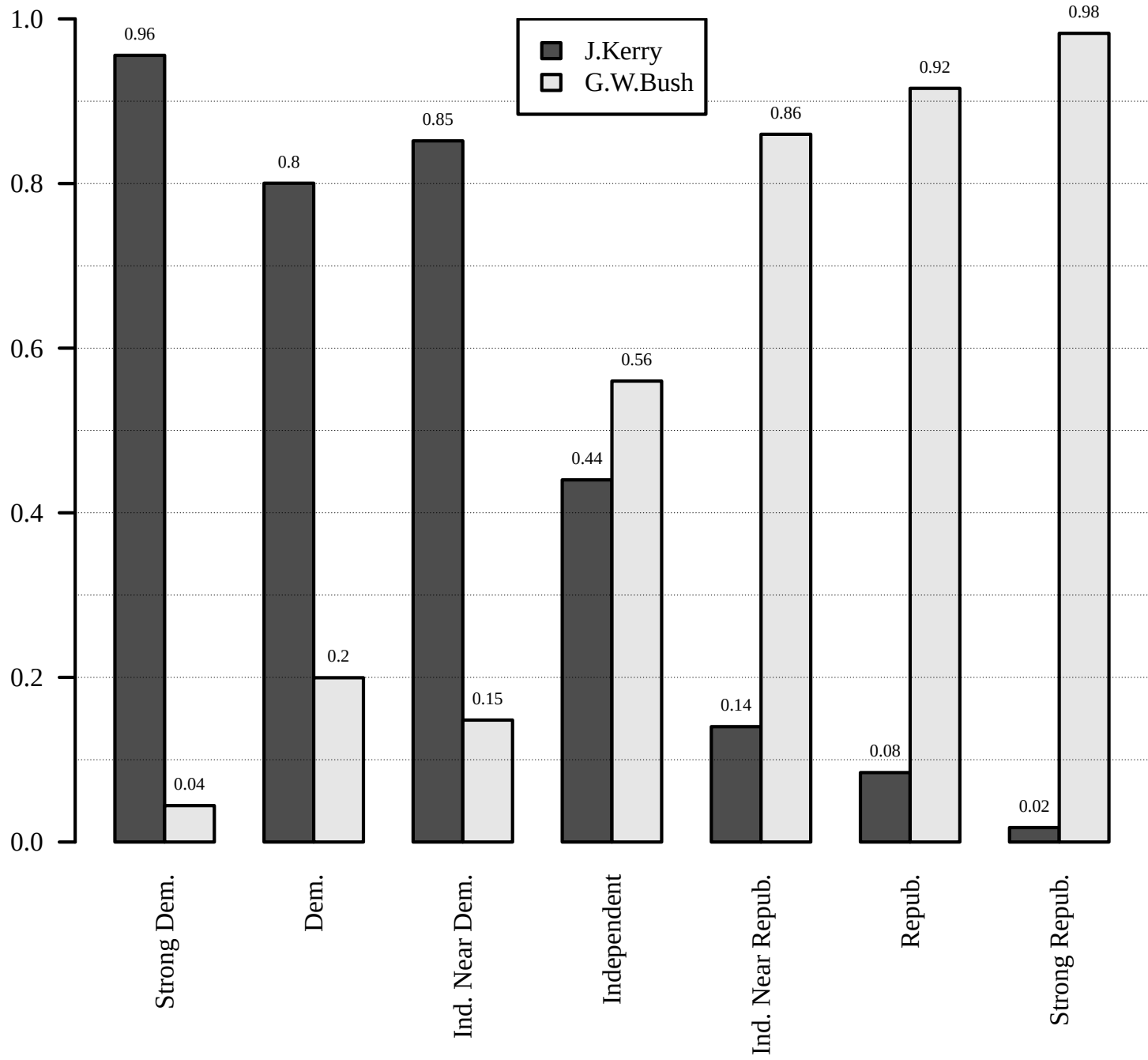
Color Coded Characters In There



Write outside the margin area

```
par(xpd=TRUE)
text(bp1,t1,label=round(t1,2), pos=3,cex=0.7,col="black")
```

- Put the Numbers on Top of the Bars (pos=3)
- par(xpd=TRUE) needed to force R to actually write outside its normal plot region.


J.Kerry
G.W.Bush
1.0
0.8
0.6
0.4
0.2
0.0
0.96
0.04
0.8
0.2
0.85
0.15
0.44
0.56
0.14
0.86
0.08
0.92
0.02
0.98
Strong Dem.
Dem.
Ind. Near Dem.
Independent
Ind. Near Repub.
Repub.
Strong Repub.