Week 4

Plotting and Color in R

better set color

grDevices:

colorRamp
colorRampPalette } interpolate to make new colors

colors() - all colors, referenced by name

colorRamp - takes a palette and returns a function that gives colors

pal = colorRamp(c("red", "blue"))

pal(0) max
255 0 0

pal(1) max
0 0 255

pal(0.5) interpolation
127.5 0 127.5

pal(seq(0, 1, len = 10))

↓

10 colors

pal <- colorRampPalette(c("red", "yellow"))

pal(2)

↓

gives just 2 colors

pal(10)

↓

10 different colors, b/c being interpolated

just returns the number of colors you ask

RColorBrewer — from CRAN

- sequential (numerical, etc from low to high)
- diverging (negative, positive, etc)
- qualitative (not ordered data)

can be used in colorRamp and colorRampPalette

```
library(RColorBrewer)
cols = brewer.pal(3, "BuGn")
cols    (3 colors)
pal = colorRampPalette
image(volcano, col = pal(20))
```

```
x = rnorm(1000)
y = rnorm(1000)
smoothScatter(x, y) — prints a hist.
```

Some other functions

rgb — returns hex string

add transparency (alpha param)

colorspace package

```
plot(x,y, col = rgb(0,0,0,0.2), pch = 19)
                    transparency
                          ↑
                          α
```

~~Regular Expressions~~

Dates and Times

Date class - only day

Time: POSIXct POSIXlt
↓ base integer ↓ uses list (day of week, etc)

x = as.Date("1970-01-01")

unclass(x) => 0 (0 days after)

Time:

weekdays, months, quarters

on POSIX*

x = Sys.time()
p = as.POSIXlt(x)

p$sec

~~strptt~~ strptime - converts dates

dsts = c("January 10, 2012 10:40"
"December 9, 2011 ...")

x = strptime(dsts, "%B %d, %Y %H:%M")

operations are applicable for dates and times:

$<$, $>$, $-$, $+$, etc

It keeps track on leap years, leap seconds, daylight savings, etc

```
x = as.Date("2012-03-01")
y = as.Date("2012-02-28")
```

x - y returns 2.

and time zones

```
x = as.POSIXct("2012-10-25 01:00:00")
y = as.POSIXct("2012-10-25 06:00:00",
               tz = "GMT")
```

x - y $\Rightarrow$ 1 hour

Reg. Exp in R

grep – returns indexes of matched
grepl – returns T/F elements

regexpr – index where match begins, first match
gregexpr – all matches
sub – replacing
gsub

homicides = readLines("homicides.txt")

↓

character vector

⇓

need to extract.

i = grep("[Cc]ause: [Ss]hooting", homicides)
j = grep("[Ss]hooting", homicide)

setdiff(i, j) => integer(0)

(everything that in i is in j)

setdiff(j, i) => 319 859

grep("^New", states, value=T)

returns values straight away

regexpr("<dd>[Ff]ound (.*?)</dd>", h[j])

↓ ~~match~~

attr "match.length"
attr "use Bytes"

⇓

substr(h, 177, 177+33+1)

regmatches

r = regexp("~~mat~~ pattern", h[1:5])

regmatches(h[1:5], r)

↑ the same as prev. code

sub("<dd>[Ff]ound at | </dd>", "t,-")

or or.

only first occurence,

↓

we need gsub. - replaces all, not only first.

(as.Date(d, "%B %d, %Y")

regexpec("__", h)

pattern

↓

List with , group!

[[1]]

177 (190)

↓ ↓

where match starts — where group starts

r = regexpr(...)

regmatches(h, r)

List with ⇓ entire match, group

Sim up:

r = regexpr("<dd>[Ff]ound on (.*?)</dd>", homicides)

m = regmatches(homicides, r)

dates = sapply(m, function(x) x[2])
dates = as.Date(dates, "%B %d, %Y")

hist(dates, "month"; freq = T)
"year"
↑
aggregate by

Classes and Methods

S3

"old-style" classes

"quick and dirty"

Separate

S4

"new-style" classes

more formal

better to use in new projects

class - description
object - instance of a class

method - operates on a certain class

generic function - for all objects
(doesn't do anything, but "redirect" requests)

?Classes ?Methods

?setClass ?setMethod ?setGeneric

S3 System!

mean - generic method

UseMethod("mean")

methods("mean")

S4

show (print for S4)

standardGeneric("show")

showMethods("show")

the first argument of a method:

1. object – gets the class (get object and looks for a method)
2. search for applicable funct
3. if a method exists, it's called
4. if doesn't, default is called
5. otherwise throws an error

```
X <- rnorm(100)
mean(X)
```

1. class of x is numeric
2. no methods for numeric!
3. default method is called

getS3method("mean", "default")

```
df = data.frame(x = rnorm(100),
                y = 1:100)
sapply(df, mean)
```

1. in each col, mean checks the class and despatches appropriate methods
2. in both cases default mean is called

Some methods are visible, but you should never call them directly!

(i.e. mean.default)

(in S4 it's not possible)

```
set.seed(10)
x = rnorm(100)
x = as.ts(x)   # convert to a time series obj
plot(x)
```

different! from just plot(rnorm)

If you create a new object, you would like to write methods for print/show, summary, plot

How to extend?

S3 — no explicit definition for classes

(S4) setClass – creates a new class

slots – data stored there

```
setClass("polygon",          <- slots
         representation(x = "numeric",
                        y = "numeric"))
```

slots for S4 objects are accessed using @

```
setMethod("plot", "polygon",
          function(x, y, ...) {
              plot(x@x, x@y, type = n, ...)
              xp <- c(x@x, x@x[1])
              yp <- c(x@y, x@y[1])
              lines(xp, yp)
          })

p = new("polygon", x = c(1, 2, 3, 4)
                   y = (1, 2, 3, 1))

plot(p)
```