Lecture 16. Defining Colors

R and Data Visualization

BIG2006, Hanyang University, Fall 2022

Representing and Using Color

Color plays a key role in many plots. It can be a critical aid to interpreting your data/models by distinguishing between values and variables.

It is useful to understand a little about how R formally represents and handles colors.

You will examine common ways to create and represent specific colors and how to define and use a cohesive collection of colors (called $palette$)

Red-Green-Blue Hexadecimal Color Codes

- The standard RGB system is assigned an integer from 0 to 255 (inclusive), can form a total of 256^3 possible colors.
- RGB *triplet*: (0,0,0) represents pure black, (255,255,255) represents pure white, and (0,255,0) is full green
- `col`: lets you select one of eight colors (1 to 8)

Note: The RGB triplets are frequently expressed as $hexadecimals$, a numeric coding system often used in computing.

In R, a hexadecimal, or $hex\ code$, is a character string with a $\#$ followed by six alphanumeric characters: the letters A through F and the digits 0 through 9.

```
palette()
```

```
## [1] "black"   "#DF536B" "#61D04F" "#2297E6" "#28E2E5" "#CD0BBC" "#F5C710"
## [8] "gray62"
```

```
col2rgb(c("black","green3","pink"))
```

```
##       [,1] [,2] [,3]
## red      0    0  255
## green    0  205  192
## blue     0    0  203
```

```
rgb(t(col2rgb(c("black","green3","pink"))),maxColorValue=255)
```

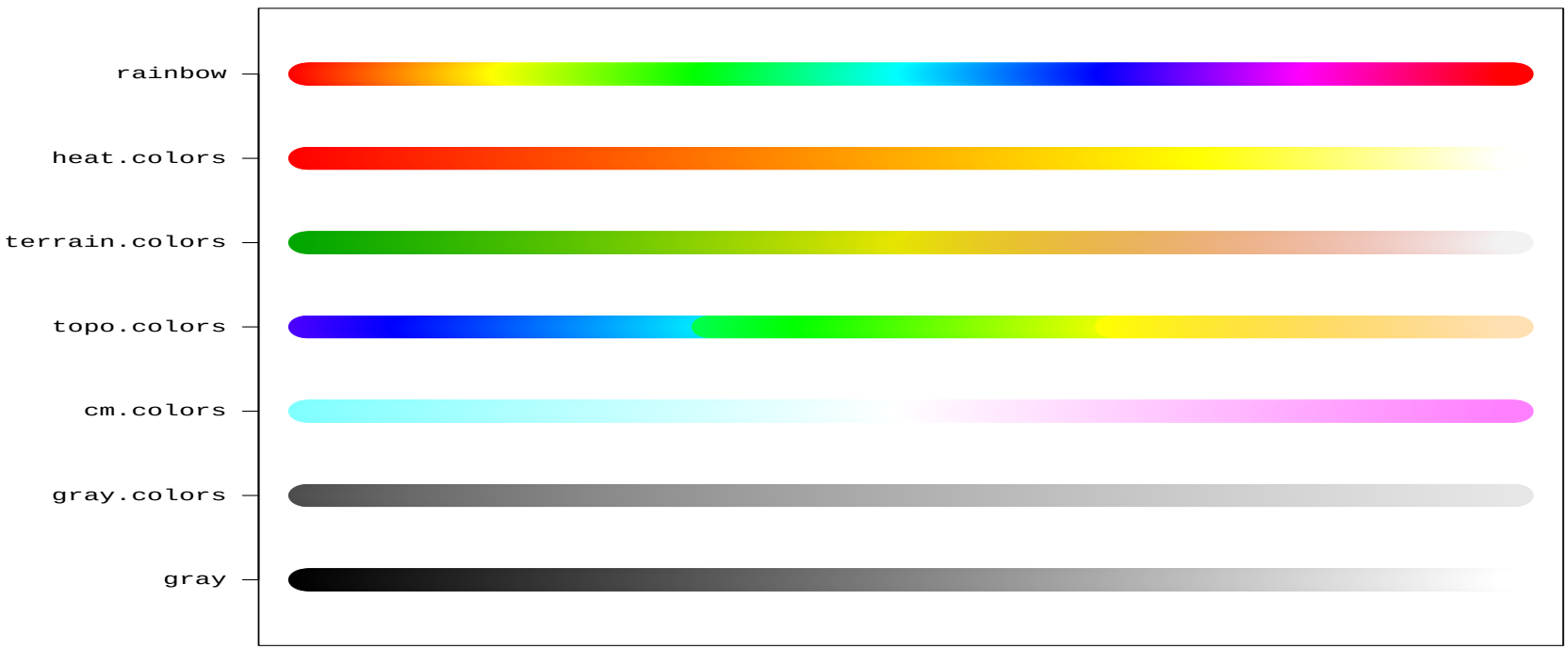
```
## [1] "#000000" "#00CD00" "#FFC0CB"
```

Built-in Palettes

- Base color palettes are defined by the functions `rainbow`, `heat.colors`, `terrain.colors`, `topo.colors`, `cm.colors`, `gray.colors`, and `gray`

```
# generate exactly 600 colors from each palette
N <- 600
rbow <- rainbow(N)
heat <- heat.colors(N)
terr <- terrain.colors(N)
topo <- topo.colors(N)
cm <- cm.colors(N)
gry1 <- gray.colors(N)
gry2 <- gray(level=seq(0,1,length=N))
```

```
par(mar=c(1,8,1,1))
plot(1,1,xlim=c(1,N),ylim=c(0.5,7.5),type="n",xaxt="n",yaxt="n",ann=FALSE)
points(rep(1:N,7),rep(7:1,each=N),pch=19,cex=3,
       col=c(rbow,heat,terr,topo,cm,gry1,gry2))
axis(2,at=7:1,labels=c("rainbow","heat.colors","terrain.colors",
                       "topo.colors","cm.colors","gray.colors","gray"),
     family="mono",las=1)
```

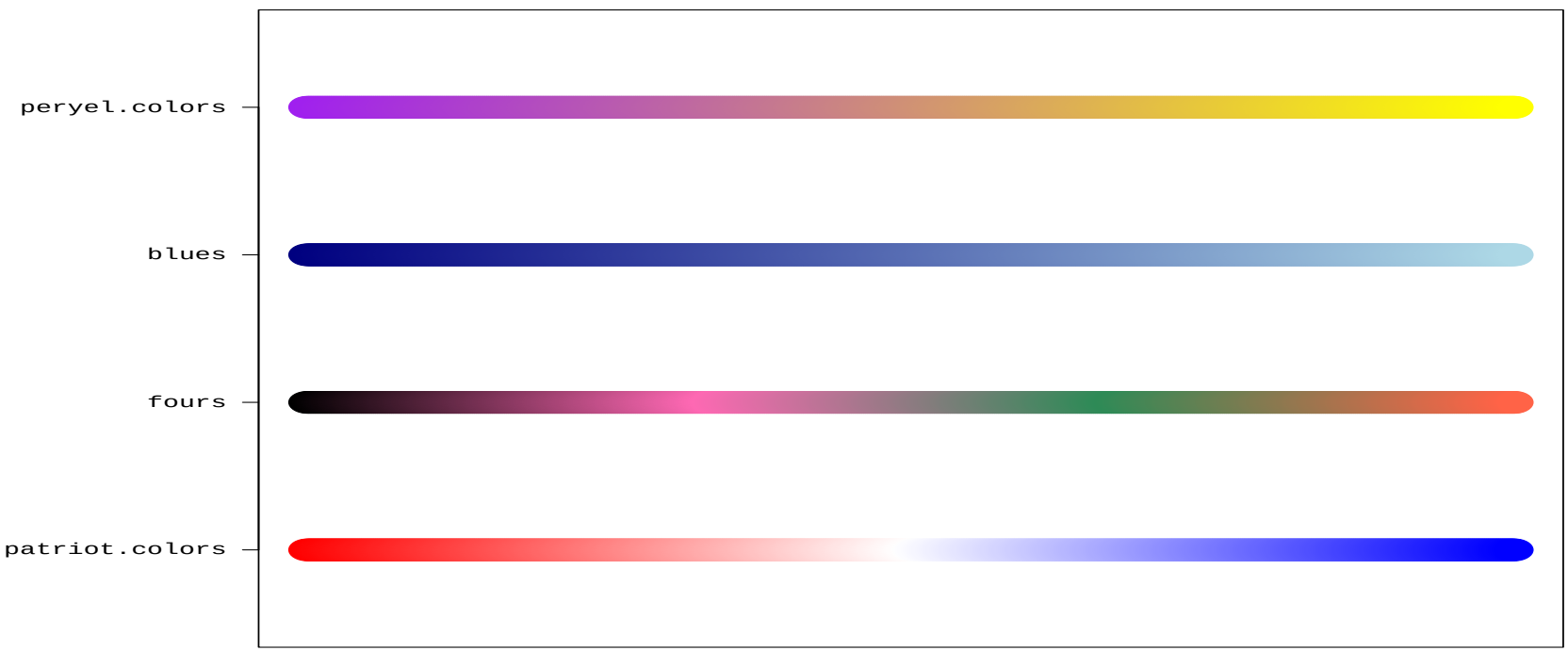


Custom Palettes

- `colorRampPalette`: allows you to create your own palettes
- You specify the key colors to be interpolated, in the desired order, as a character vector of names from the collection that R recognizes.

```
puryel.colors <- colorRampPalette(colors=c("purple","yellow"))
blues <- colorRampPalette(colors=c("navyblue","lightblue"))
# using more than two colors
fours <- colorRampPalette(colors=c("black","hotpink","seagreen4","tomato"))
patriot.colors <- colorRampPalette(colors=c("red","white","blue"))
```

```
py <- puryel.colors(N)
bls <- blues(N)
frs <- fours(N)
pat <- patriot.colors(N)
par(mar=c(1,8,1,1))
plot(1,1,xlim=c(1,N),ylim=c(0.5,4.5),type="n",xaxt="n",yaxt="n",ann=FALSE)
points(rep(1:N,4),rep(4:1,each=N),pch=19,cex=3,col=c(py,bls,frs,pat))
axis(2,at=4:1,labels=c("peryel.colors","blues","fours","patriot.colors"),
     family="mono",las=1)
```



Using Color Palettes to Index a Continuum

- Color can be used to identify groups based on a categorical variable.
- Assigning colors appropriately to values on a continuum requires a little more thought.
- There are two methods: through categorization or through normalization of your continuous values.

Method 1. Via Categorization

- Bin your continuous values into a fixed number of k categories
- Generate k colors from your palette
- Match each observation to the appropriate color based on the bin it falls into

```r
surv <- na.omit(survey[,c("Wr.Hnd","NW.Hnd","Height")])
NW.pal <- colorRampPalette(colors=c("red4","yellow2"))
k <- 5; ryc <- NW.pal(k); ryc
```

```
## [1] "#8B0000" "#A33B00" "#BC7700" "#D5B200" "#EEEE00"
```

```r
NW.breaks <- seq(min(surv$NW.Hnd),max(surv$NW.Hnd),length=k+1)
NW.breaks
```

```
## [1] 12.5 14.7 16.9 19.1 21.3 23.5
```

```r
NW.fac <- cut(surv$NW.Hnd,breaks=NW.breaks,include.lowest=TRUE)
as.numeric(NW.fac)[1:14]
```

```
##  [1] 3 4 3 4 3 3 3 4 3 3 2 4 3 3
```

```r
NW.cols <- ryc[as.numeric(NW.fac)]
NW.cols[1:14]
```

```
##  [1] "#BC7700" "#D5B200" "#BC7700" "#D5B200" "#BC7700" "#BC7700" "#BC7700"
##  [8] "#D5B200" "#BC7700" "#BC7700" "#A33B00" "#D5B200" "#BC7700" "#BC7700"
```

Method 2. Via Normalization

- You need to provide a numeric vector of values to tell R, on a continuous scale from 0 through 1, how "far along" the palette to go.
- `colorRamp`: allows you to create your palette used in the same way as `colorRampPalette`, but the result is a color palette function that expects a numeric vector.
- To transform a collection of n original values $\{x_1, \ldots, x_n\}$ to $\{z_1, \ldots, z_n\}$ where $0 \leq z_i \leq 1$, you employ the following:

$$z_i = \frac{x_i - \min x_i}{\max x_i - \min x_i}.$$

```r
normalize <- function(datavec){
  lo <- min(datavec,na.rm=TRUE)
  up <- max(datavec,na.rm=TRUE)
  datanorm <- (datavec-lo)/(up-lo)
  return(datanorm)
}
surv$NW.Hnd[1:14]
```

```
##  [1] 18.0 20.5 18.9 20.0 17.7 17.7 17.3 19.5 18.5 17.2 16.0 20.2 17.0 18.0
```

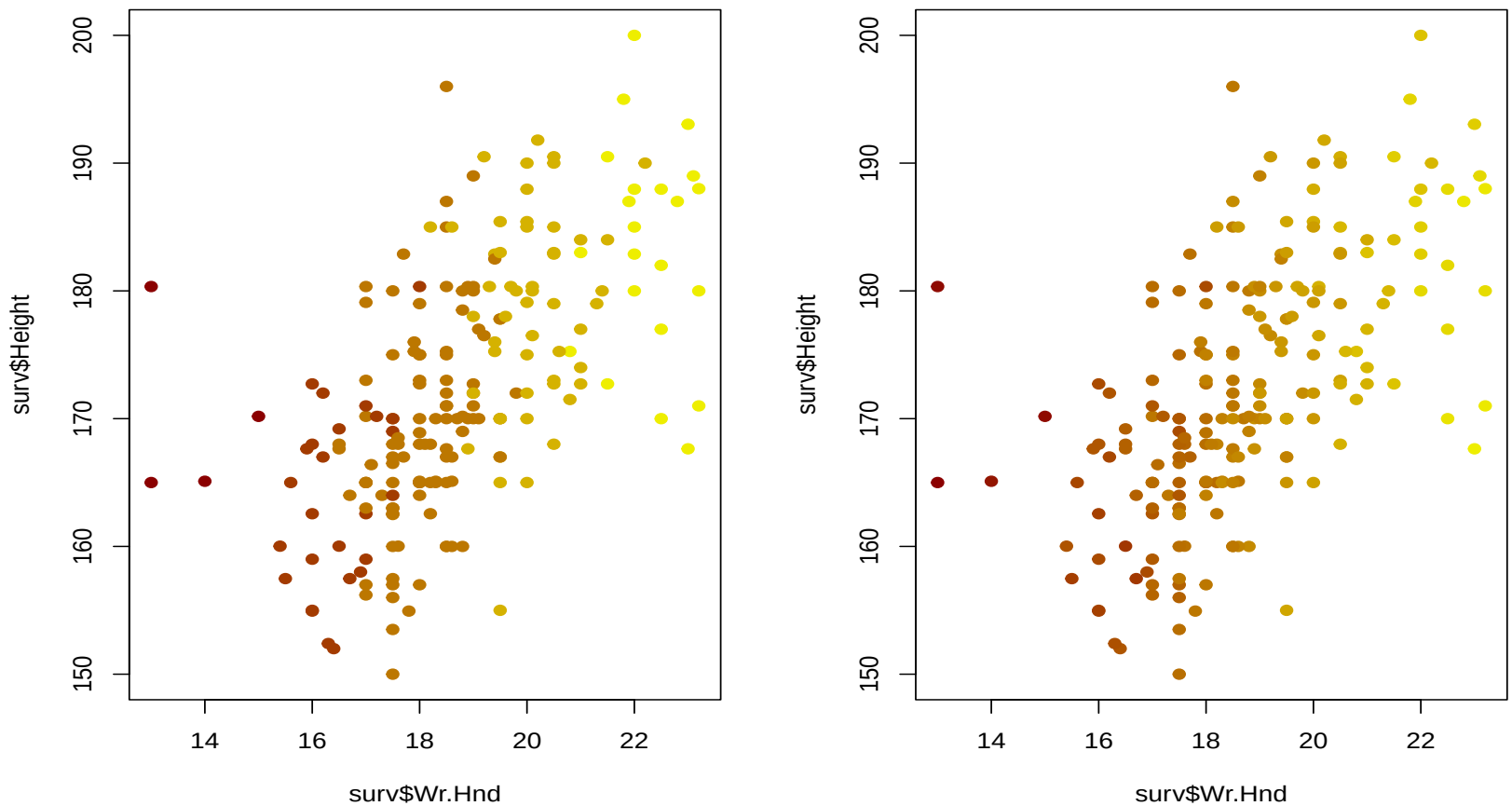
```r
normalize(surv$NW.Hnd)[1:14]
```

```
##  [1] 0.5000000 0.7272727 0.5818182 0.6818182 0.4727273 0.4727273 0.4363636
##  [8] 0.6363636 0.5454545 0.4272727 0.3181818 0.7000000 0.4090909 0.5000000
```

```r
NW.pal2 <- colorRamp(colors=c("red4","yellow2"))
ryc2 <- NW.pal2(normalize(surv$NW.Hnd))
NW.cols2 <- rgb(ryc2,maxColorValue=255)
NW.cols2[1:14]
```

```
##  [1] "#BC7700" "#D3AD00" "#C48A00" "#CEA200" "#B97000" "#B97000" "#B66700"
##  [8] "#CA9700" "#C18100" "#B56500" "#AA4B00" "#D0A600" "#B36100" "#BC7700"
```

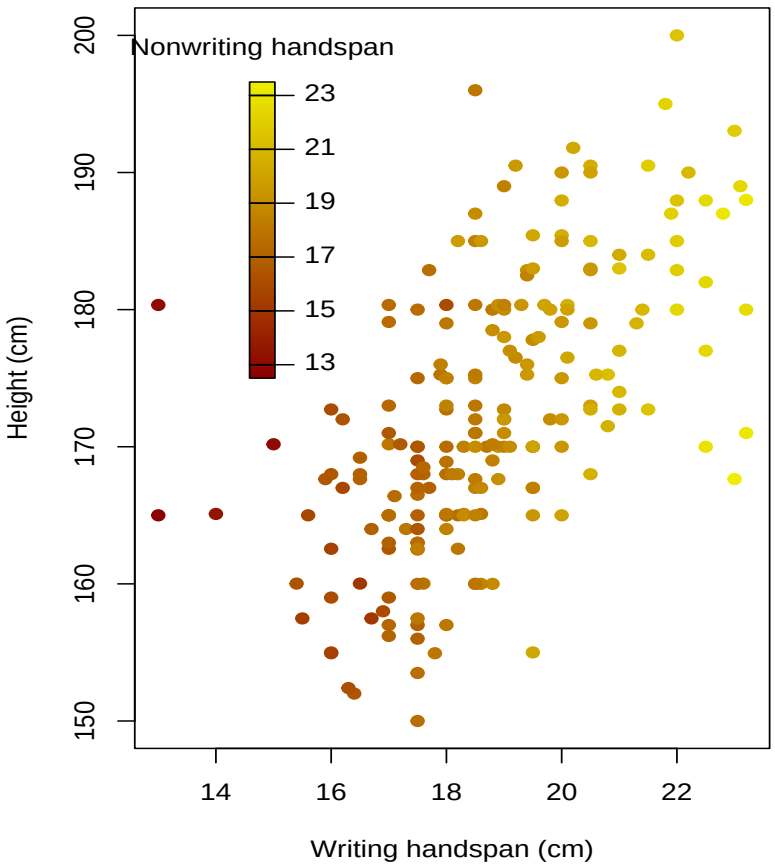
```
par(mfrow=c(1,2))
plot(surv$Wr.Hnd,surv$Height,col=NW.cols,pch=19) # categorization
plot(surv$Wr.Hnd,surv$Height,col=NW.cols2,pch=19) # normalization
```

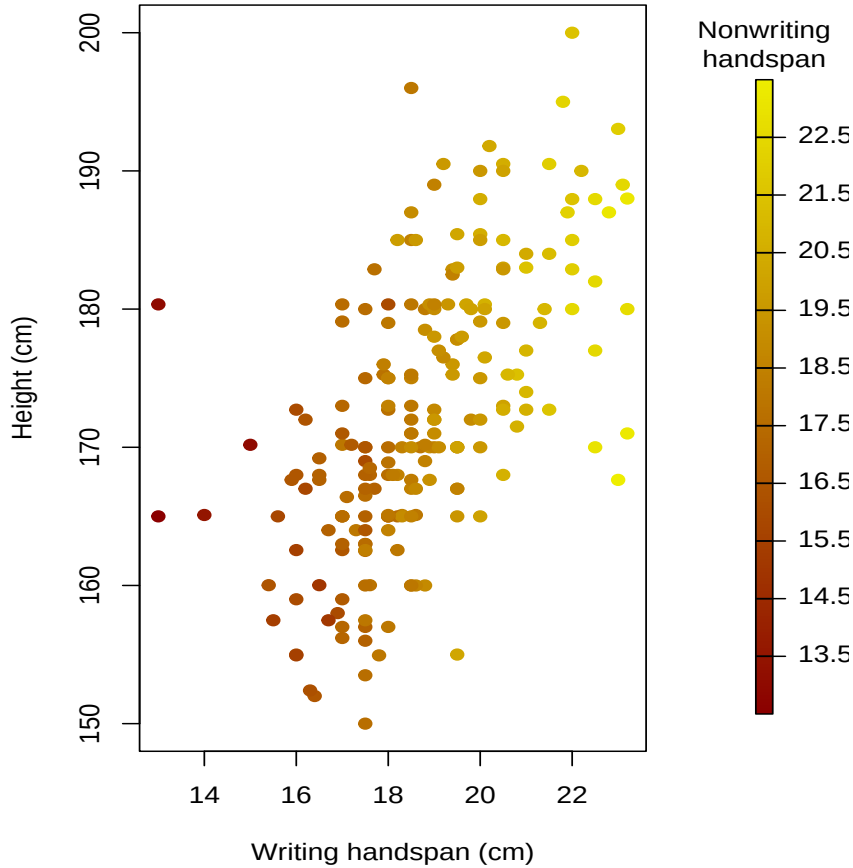


Including a Color Legend

- `colorlegend` in the `shape` package
- The positioning and sizing of the color legend: in *Relative device coordinates*, horizontal (`posx`) and vertical (`posy`) lengths of the strip

```
plot(surv$Wr.Hnd,surv$Height,col=NW.cols2,pch=19,
     xlab="Writing handspan (cm)",ylab="Height (cm)")
colorlegend(NW.pal(200),zlim=range(surv$NW.Hnd),zval=seq(13,23,by=2),
            posx=c(0.3,0.33),posy=c(0.5,0.9),main="Nonwriting handspan")
par(mar=c(5,4,4,6))
plot(surv$Wr.Hnd,surv$Height,col=NW.cols2,pch=19,
     xlab="Writing handspan (cm)",ylab="Height (cm)")
colorlegend(NW.pal(200),zlim=range(surv$NW.Hnd),zval=13.5:22.5,digit=1,
            posx=c(0.89,0.91),main="Nonwriting\nhandspan")
```


Nonwriting handspan
23
21
19
17
15
13
Height (cm)
150
160
170
180
190
200
14
16
18
20
22
Writing handspan (cm)


Nonwriting
handspan
22.5
21.5
20.5
19.5
18.5
17.5
16.5
15.5
14.5
13.5
Height (cm)
150
160
170
180
190
200
14
16
18
20
22
Writing handspan (cm)

Opacity

- All colors and color palettes functions that provide the user with hex codes have an optional argument `alpha`.
- After the #, eight will appear, with the last two containing the additional opacity information.

```
rgb(cbind(255,0,0),maxColorValue=255,alpha=0) # zero opacity
```

```
## [1] "#FF000000"
```

```
rgb(cbind(255,0,0),maxColorValue=255,alpha=102) # 40 percent opacity (0.4*255)
```

```
## [1] "#FF000066"
```

```
rgb(cbind(255,0,0),maxColorValue=255,alpha=255) # full opacity
```

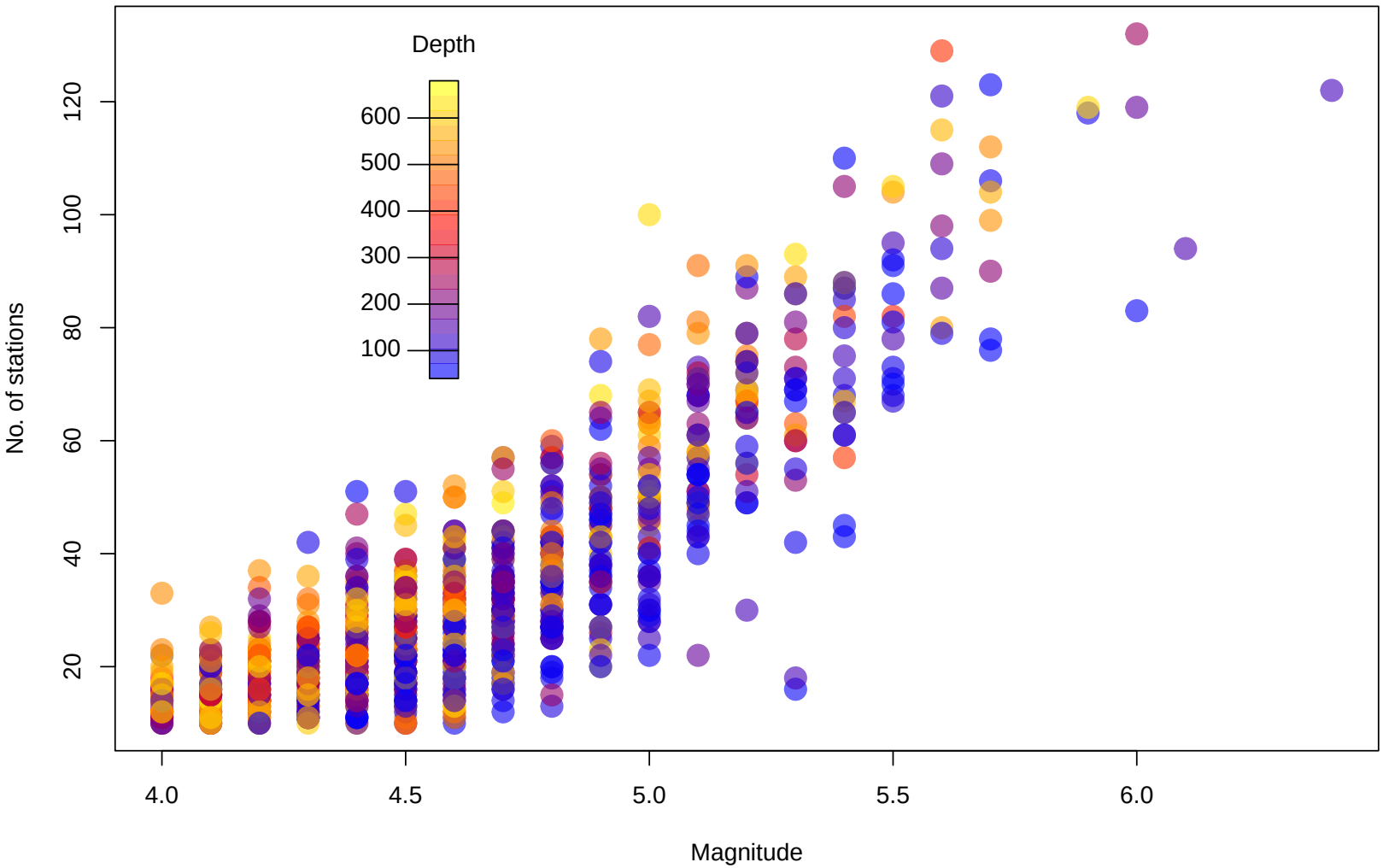
```
## [1] "#FF0000FF"
```

- You can adjust the opacity of any color you've already got with the `alpha.f` argument of `adjustcolor` function.

```
adjustcolor(rgb(cbind(255,0,0),maxColorValue=255),alpha.f=0.4)
```

```
## [1] "#FF000066"
```

```
# quakes data frame (1000 seismic events near Fiji)
keycols <- c("blue","red","yellow")
depth.pal <- colorRampPalette(keycols)
depth.pal2 <- colorRamp(keycols)
depth.cols <- rgb(depth.pal2(normalize(quakes$depth)),maxColorValue=255,
                  alpha=0.6*255)
plot(quakes$mag,quakes$stations,pch=19,cex=2,col=depth.cols,
     xlab="Magnitude",ylab="No. of stations")
colorlegend(adjustcolor(depth.pal(20),alpha.f=0.6),
            zlim=range(quakes$depth),zval=seq(100,600,100),
            posx=c(0.3,0.32),posy=c(0.5,0.9),left=TRUE,main="Depth")
```


Depth
600
500
400
300
200
100
No. of stations
20
40
60
80
100
120
4.0
4.5
5.0
5.5
6.0
Magnitude

RGB Altenatives and Further Functionality

Other specifications include hue-saturation-value (HSV) and hue-chroma-luminance (HCL), available through the built-in `hsv` and `hcl` functions.

Contributed functionality offers even more flexibility. For example, `RColorBrewer` packages provides more options for creating palettes than are supplied by the built-in functionality `colorRampPalette` and `colorRamp`.

Reference

- Davies, T. M. The Book of R. No Starch Press. Chapter 25.