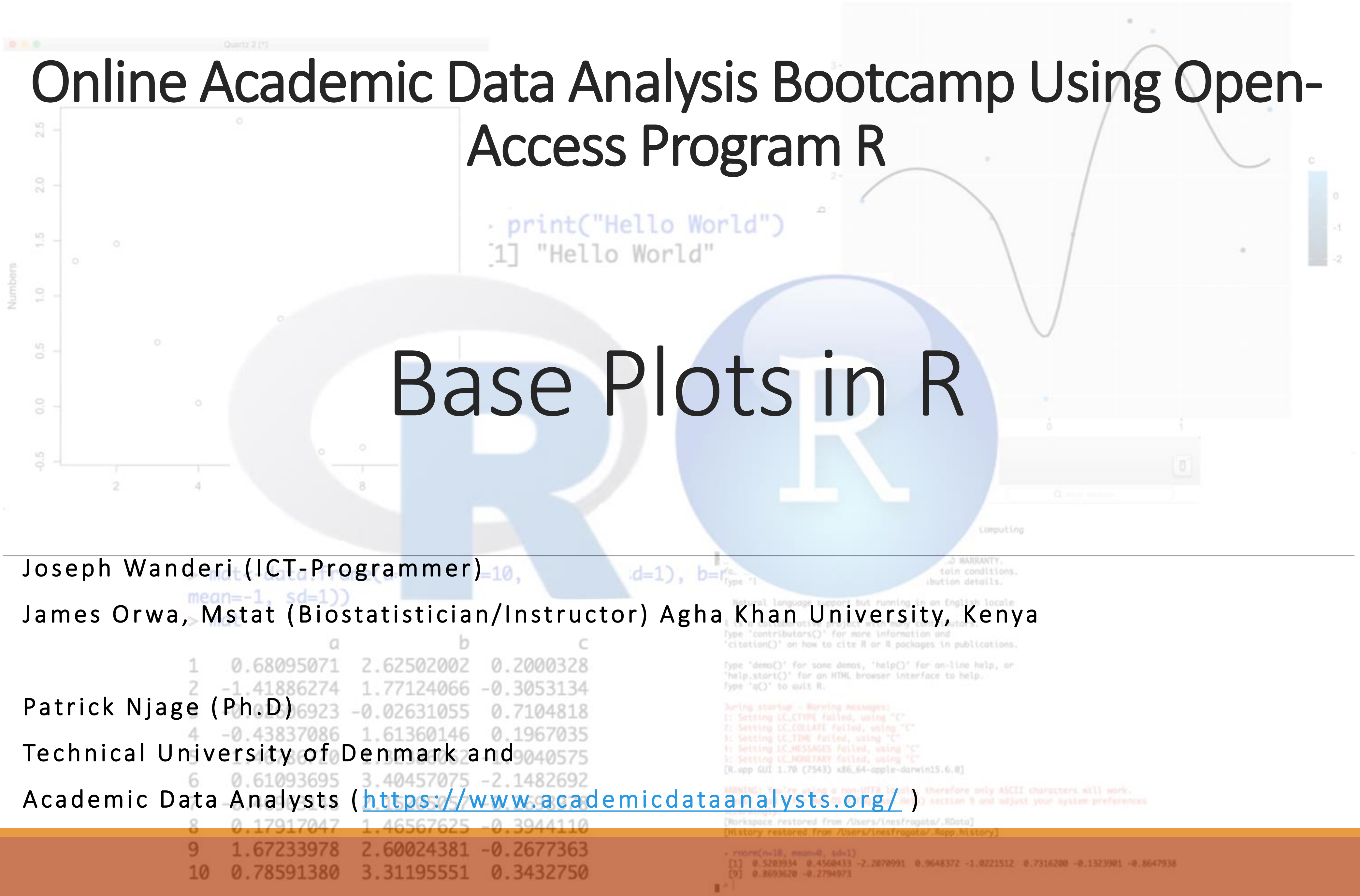


# Online Academic Data Analysis Bootcamp Using Open-Access Program R

# Base Plots in R

Joseph Wanderi (ICT-Programmer)

James Orwa, Mstat (Biostatistician/Instructor) Agha Khan University, Kenya

Patrick Njage (Ph.D)

Technical University of Denmark and

Academic Data Analysts ( https://www.academicdataanalysts.org/ )

Creating a Graph

Histograms and Density Plots

Dot Plots

Bar Plots

Line Charts

Pie Charts

Boxplots

Scatterplots

Advanced Graphics: graphical parameters, axes and text, combining plots

# Introduction: generic plot types in R

**plot()** function is the generic function for **plotting** in R. It can be used to create basic **graphs**.

A simplified format of the function is

plot(x, y, type="p")

**x** and **y**: the coordinates of points to plot

**type** : the type of graph to create; Possible values are :

type="p": for **p**oints (by default)

type="l": for **l**ines

type="b": for **b**oth; points are connected by a line

type="o": for both '**o**verplotted';

type="h": for '**h**istogram' like vertical lines

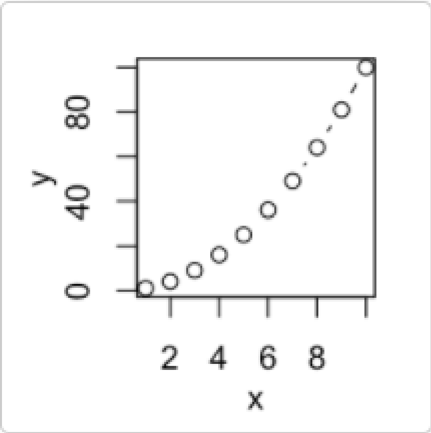
type="s": for **s**tair steps

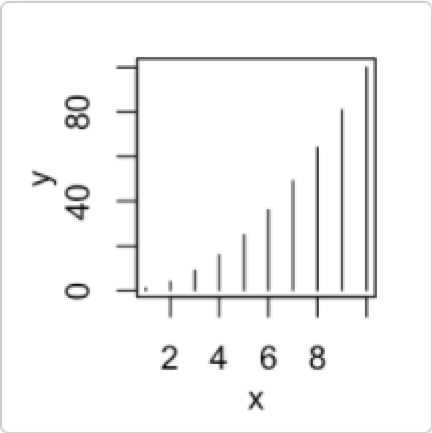
type="n": for **n**o plotting

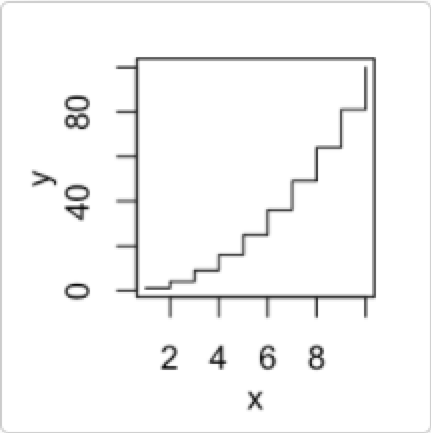
# Introduction: generic plot types in R

Examples

```
x<-1:10;

y=x*x

plot(x, y, type="b")

plot(x, y, type="h")

plot(x,y, type="s")
```







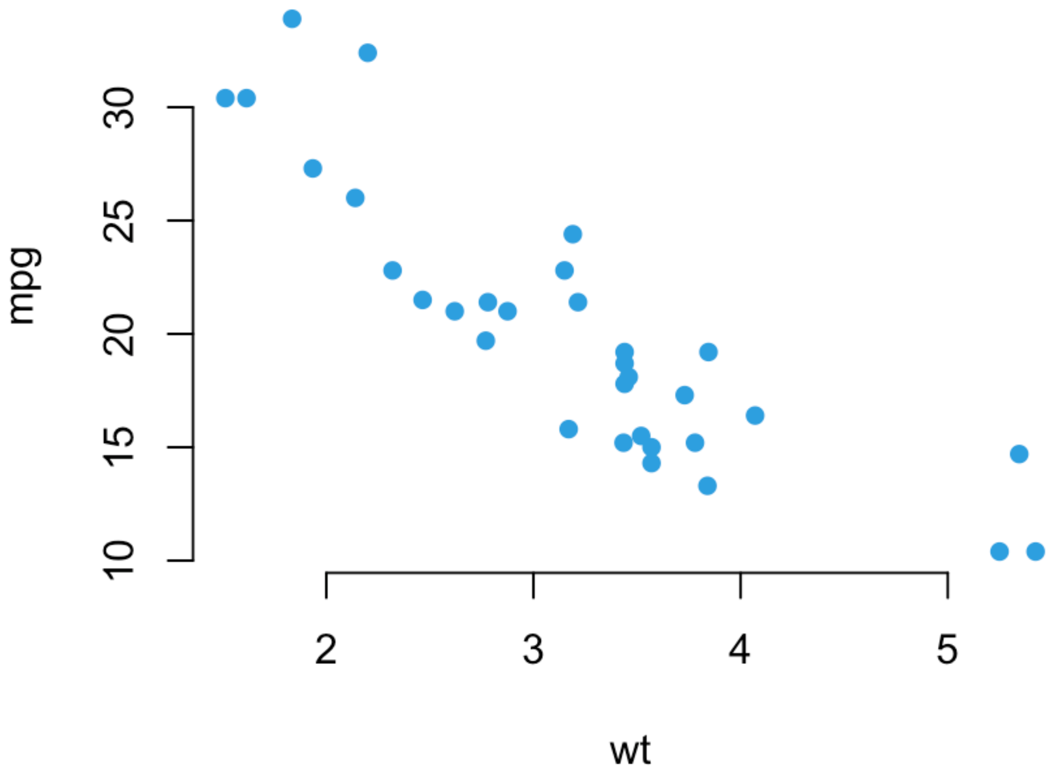
# Creating a Graph

The R base function **plot**() can be used to create graphs.

In R, graphs are typically created interactively.

```
# Creating a Graph
plot(x = mtcars$wt, y = mtcars$mpg,
pch = 16, frame = FALSE, xlab = "wt",
ylab = "mpg", col = "#2E9FDF")
```

The plot( ) function opens a graph window and plots weight vs. miles per gallon.

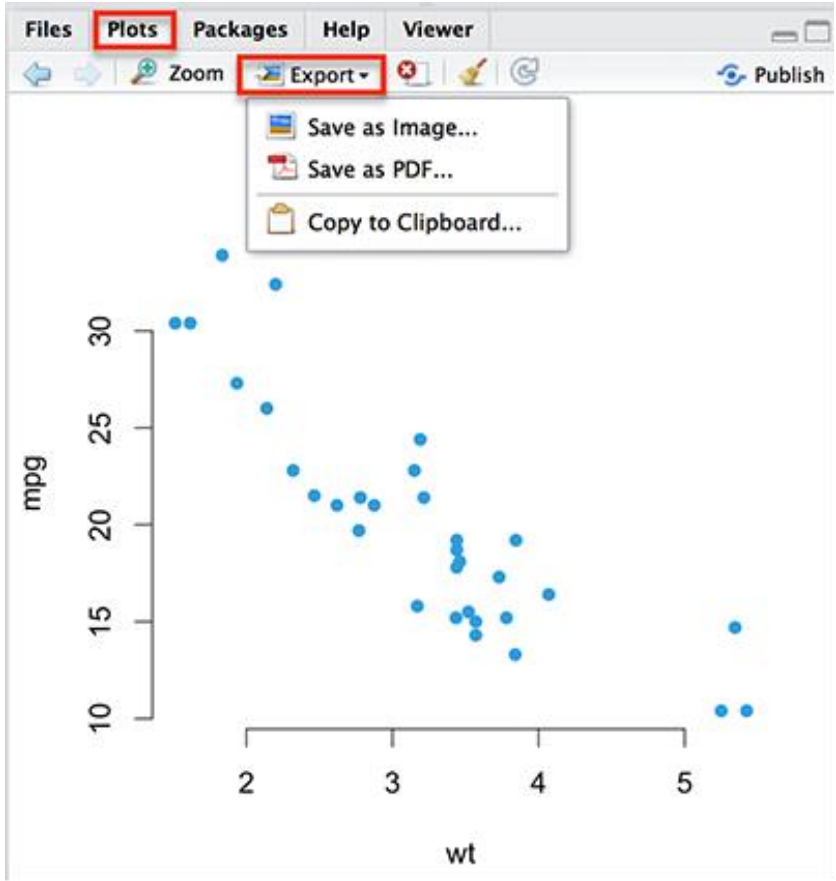


# Saving graphs: RStudio Plots Panel

If you are working with RStudio, the plot can be exported from menu in plot panel (lower right-pannel).

**Plots panel –> Export –> Save as Image or Save as PDF**

The choose directory and change file name



# Saving graphs: R codes to redirect graphs

It’s also possible to save the graph using R codes as follow:

1. Specify files to save your image using a function such as **jpeg**(), **png**(), **svg**() or **pdf**(). Additional argument indicating the width and the height of the image can be also used.

2. Create the plot

3. Close the file with **dev.off**()

Example 1: saving as pdf

```
# 1. Open a pdf file

 pdf("rplot.pdf")

# 2. Create a plot

plot(x = mtcars$wt, y = mtcars$mpg, pch = 16, frame = FALSE, xlab = "wt", ylab = "mpg", col = "#2E9FDF")

# 3. Close the pdf file

dev.off()
```

# Saving graphs: R codes to redirect graphs

Example 2: saving as jpeg file

```
# 1. Open jpeg file
jpeg("rplot.jpg", width = 350, height = "350")
 # 2. Create the plot
plot(x = mtcars$wt, y = mtcars$mpg, pch = 16, frame = FALSE, xlab = "wt", ylab = "mpg", col = "#2E9FDF")
# 3. Close the file
dev.off()
```

The R codes in the previous 2 slides saves the files in the current working directory.

# Saving graphs: R codes to redirect graphs

To redirect graphic output use one of the following functions. Use **dev.off( )** to return output to the terminal.

| **Function** | **Output to** |
|---|---|
| pdf("mygraph.pdf") | pdf file |
| win.metafile("mygraph.wmf") | windows metafile |
| png("mygraph.png") | png file |
| jpeg("mygraph.jpg") | jpeg file |
| bmp("mygraph.bmp") | bmp file |
| postscript("mygraph.ps") | postscript file |

# Create histogram plots: hist()

Histograms display the distribution of a continuous variable by dividing up the range of scores into a specified number of bins on the x-axis and displaying the frequency of scores in each bin on the y-axis.

A histogram can be created using the function hist(), which simplified format is as follow:

hist(x, breaks = "Sturges")

Where:

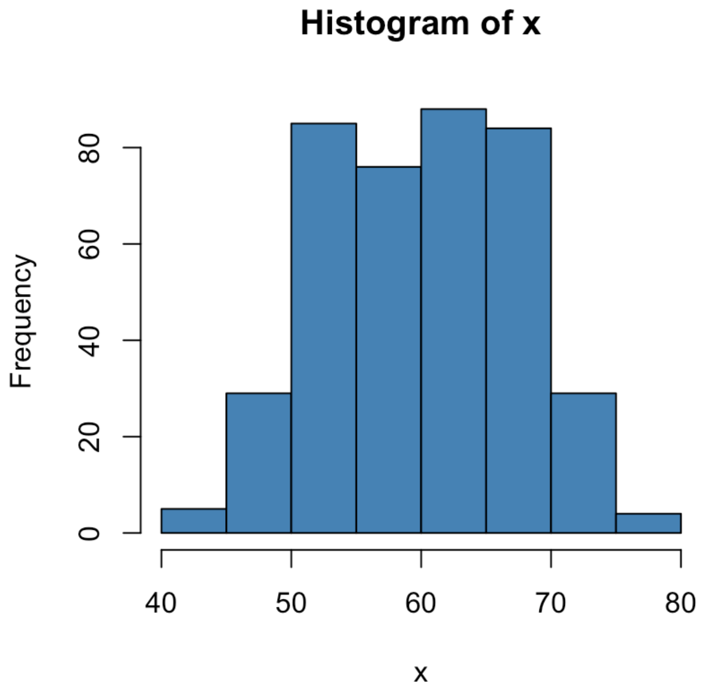
**x**: a is numeric vector

**breaks**: breakpoints between histogram cells

Example:

x <- mtcars$mpg

hist(x, col = "steelblue", frame = FALSE)

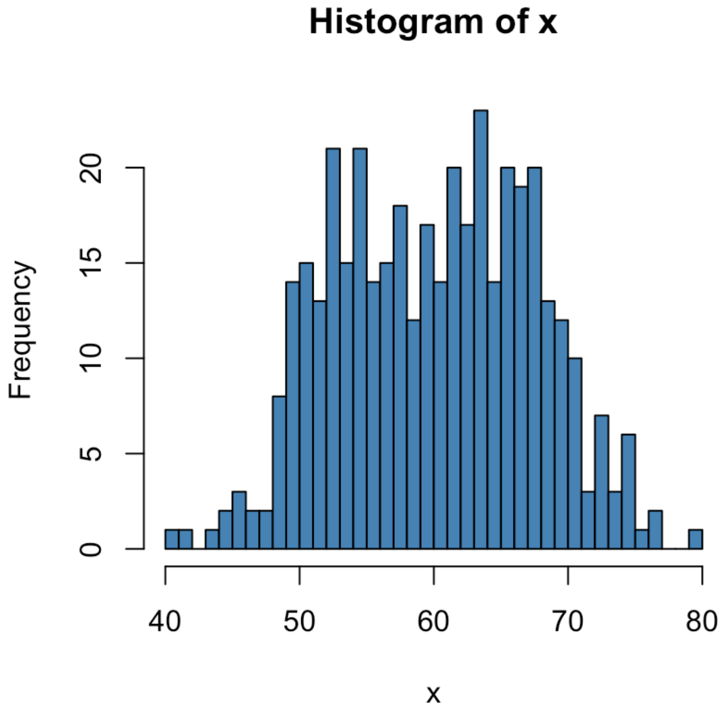


# Create histogram plots: hist()

# Change the number of breaks

```
hist(x, col = "steelblue", frame = FALSE, breaks = 30)
```

Histograms can be a poor method for determining the shape of a distribution – they are strongly affected by the number of bins used.



# Kernel Density Plots: density()

**Kernel density plots** are usually a much more effective than **histograms** way to view the distribution of a variable.

Kernel density estimation is a nonparametric method for estimating the probability density function of a random variable.

The function **density**() is used to estimate kernel density.

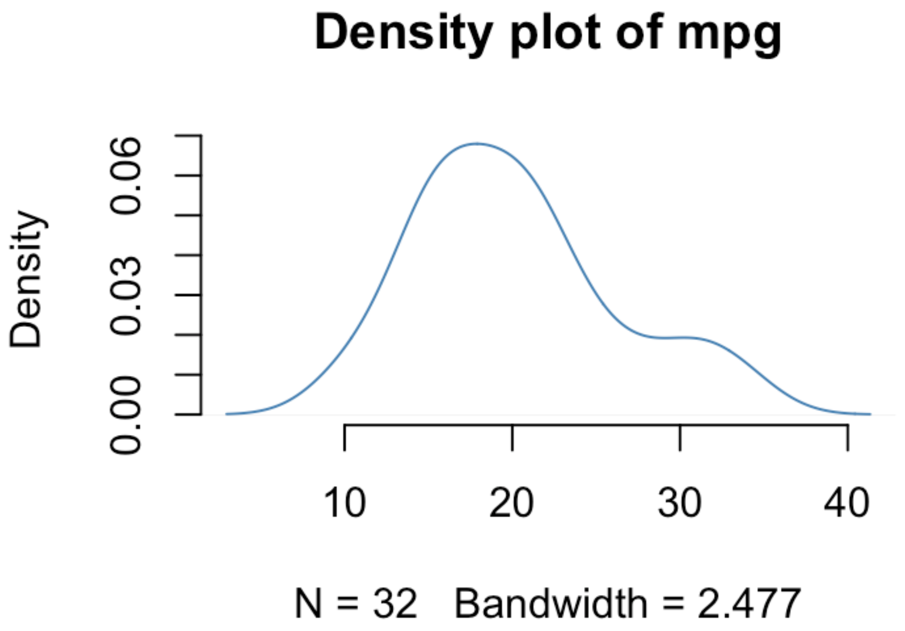
Create the plot using **plot(density(*x*))** where *x* is a numeric vector.

# Compute the density data

```
dens <- density(mtcars$mpg)
```

# plot density

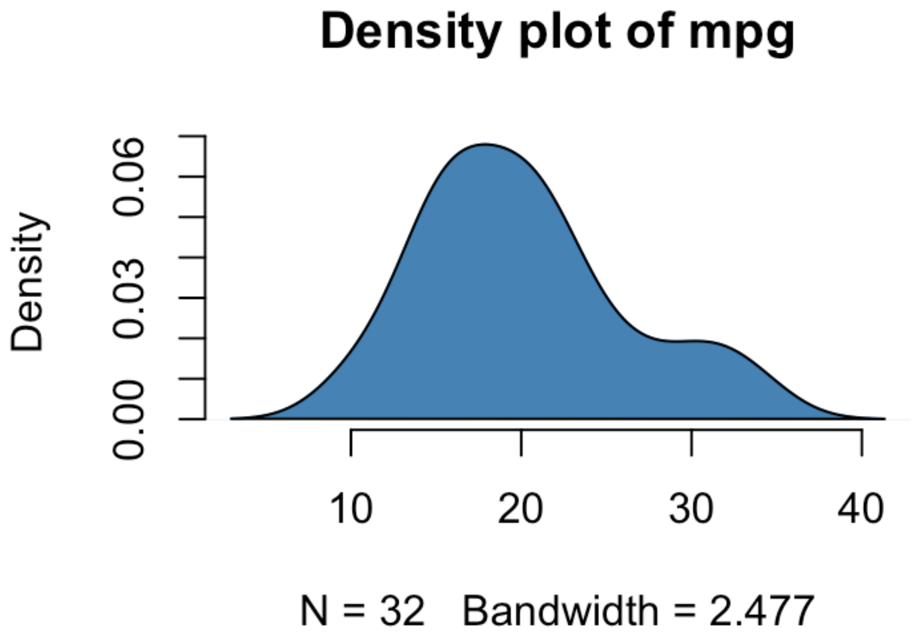
```
plot(dens, frame = FALSE, col = "steelblue", main =
"Density plot of mpg")
```



# Kernel Density Plots: density()

# Fill the density plot using polygon()

```
plot(dens, frame = FALSE, col = "steelblue",
main = "Density plot of mpg")

polygon(dens, col = "steelblue")
```



# Dot Plots: dotchart()

The function **dotchart**() is used to draw a cleveland dot plot.

dotchart(x, labels = NULL, groups = NULL, gcolor = par("fg"), color = par("fg"))

**x**: numeric vector or matrix

**labels**: a vector of labels for each point.

**groups**: a grouping variable indicating how the elements of x are grouped.

**gcolor**: color to be used for group labels and values.

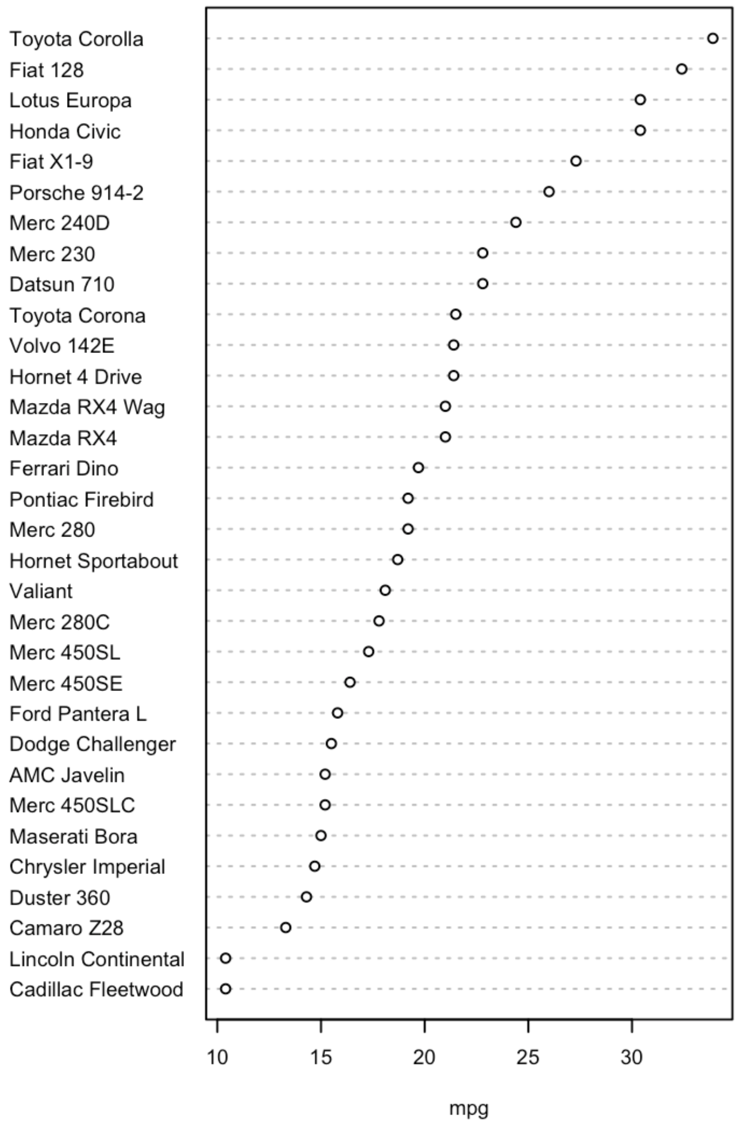
**color**: the color(s) to be used for points and labels.

**cex:** controls the size of the labels.

# Dot Plots: dotchart()

## Dot chart of one numeric vector

```
# Dot chart of a single numeric vector

dotchart(mtcars$mpg, labels =
row.names(mtcars),  cex = 0.6, xlab = "mpg")
```



# Dot Plots: dotchart()

**Dot chart of one numeric**

Plot and color by groups cyl

#Groups

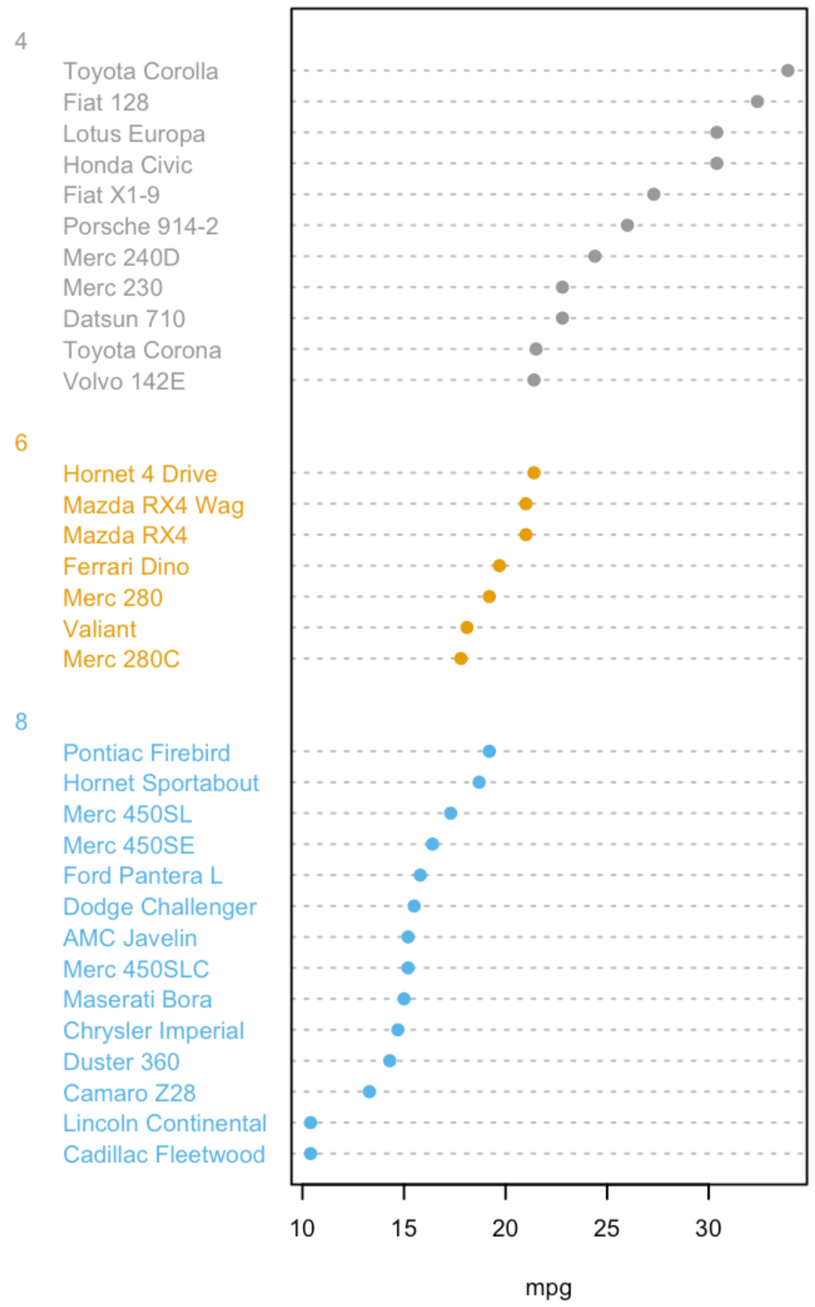
grps <- as.factor(mtcars$cyl)

#Colours for each group

my_cols <- c("#999999", "#E69F00", "#56B4E9")

#Plot

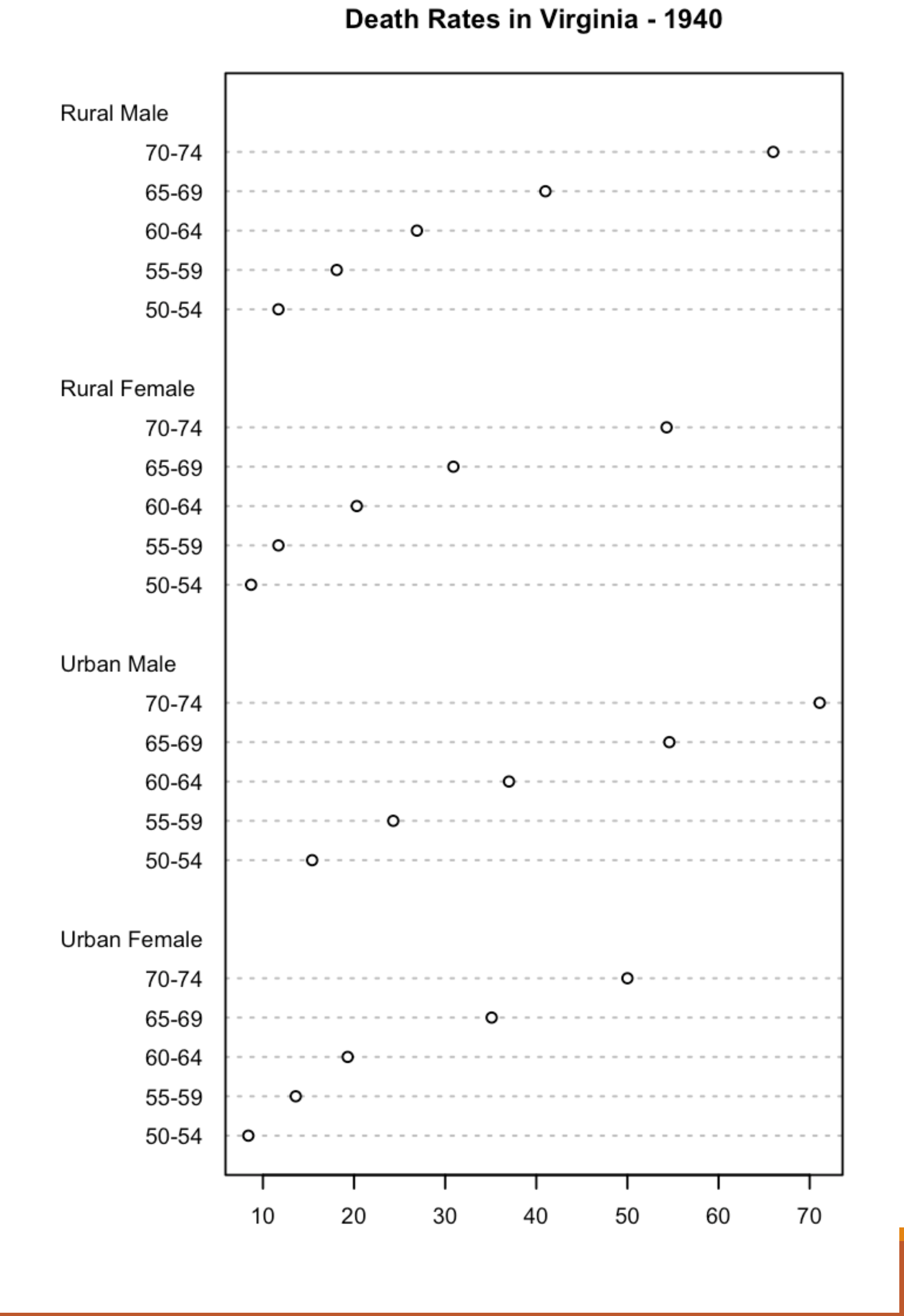
dotchart(mtcars$mpg, labels = row.names(mtcars), groups = grps, gcolor = my_cols, color = my_cols[grps], cex = 0.6, pch = 19, xlab = "mpg")



# Dot Plots: dotchart()

**Dot chart of a matrix**

```
dotchart(VADeaths, cex = 0.6, main = "Death Rates in Virginia - 1940")
```



# Bar Plots

Create barplots with the **barplot(***height***)** function

***height* is a vector or matrix**

If **height is a vector:** the values determine the heights of the bars in the plot.

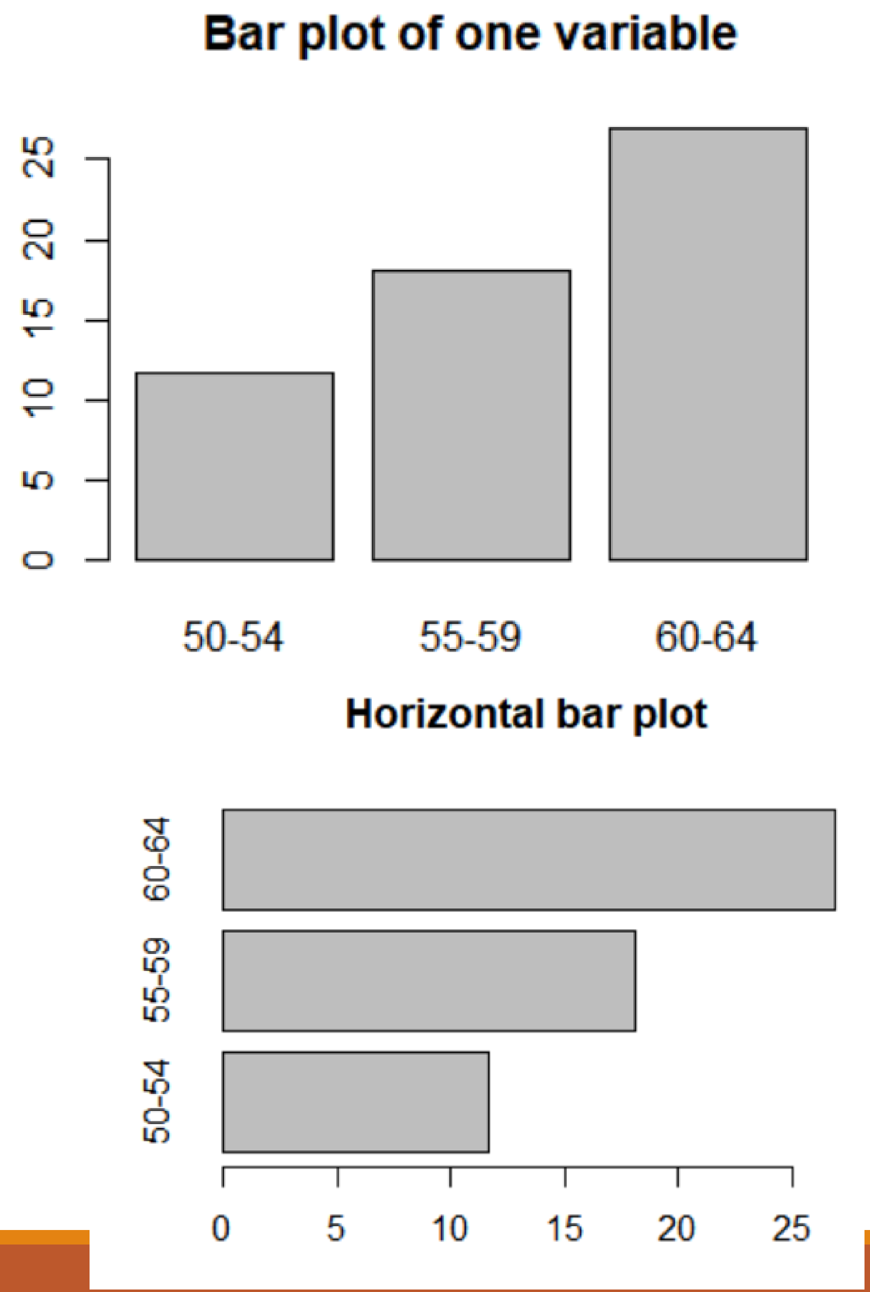
If **height is a matrix** and the option **beside=FALSE** then each bar of the plot corresponds to a column of height, with the values in the column giving the heights of stacked “sub-bars”.
If **height is a matrix** and **beside=TRUE**, the values in each column are juxtaposed rather than stacked

**names.arg=(***character vector***)** labels the bars

**horiz=TRUE** creates a horizontal barplot

# Bar Plots:basic

```
# Subset
x <- VADeaths[1:3, "Rural Male"]
x
## 50-54 55-59 60-64
##  11.7  18.1  26.9
# Bar plot of one variable
barplot(x)
# Horizontal bar plot
barplot(x, horiz = TRUE)
```

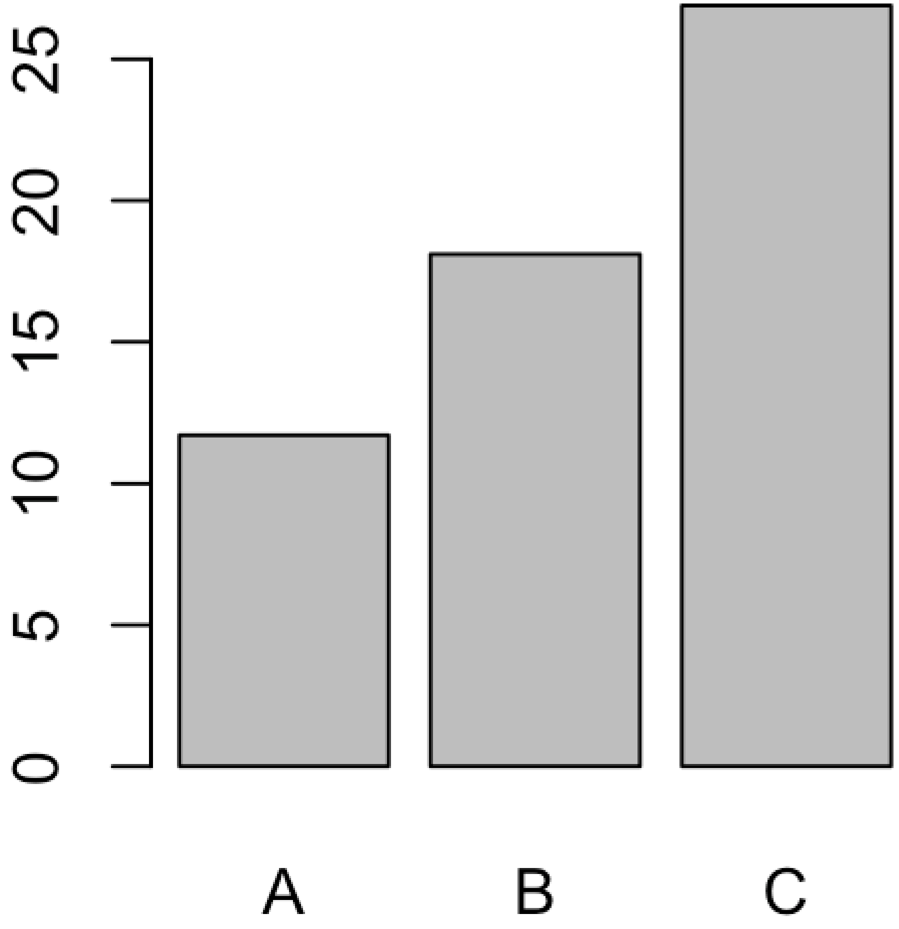

Bar plot of one variable

Horizontal bar plot

# Bar Plots

## Change group names

```
barplot(x, names.arg = c("A", "B", "C"))
```



# Bar Plots

**Change color**

**# Change border and fill color using one single color**

```
barplot(x, col = "white", border = "steelblue")
```

**#  Use different colors for each group**

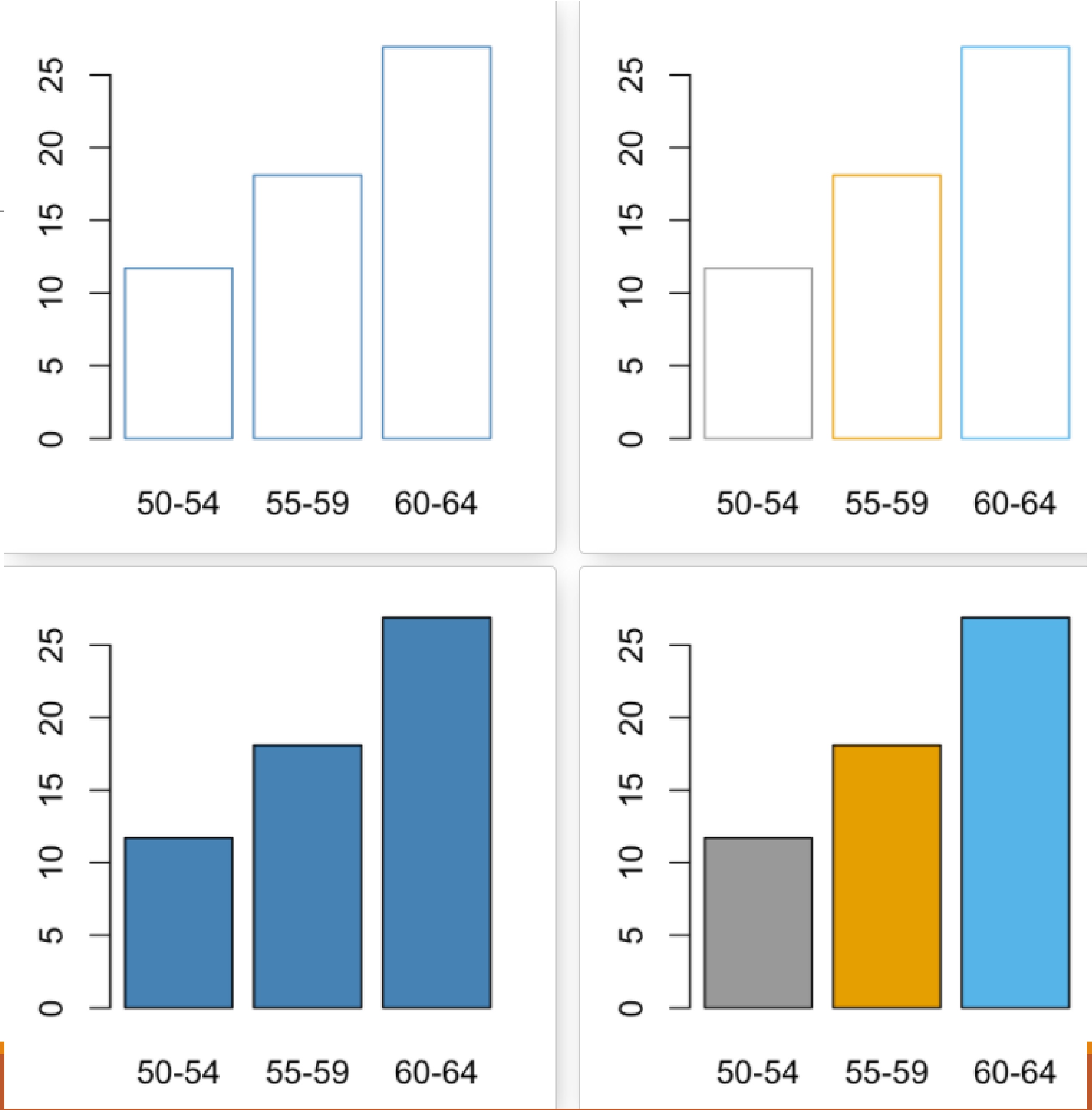
```
barplot(x, col = "white",
      border = c("#999999", "#E69F00", "#56B4E9"))
```

**# Change fill color : single color**

```
barplot(x, col = "steelblue")
```

**# Change fill color: multiple colors**

```
barplot(x, col = c("#999999", "#E69F00", "#56B4E9"))
```



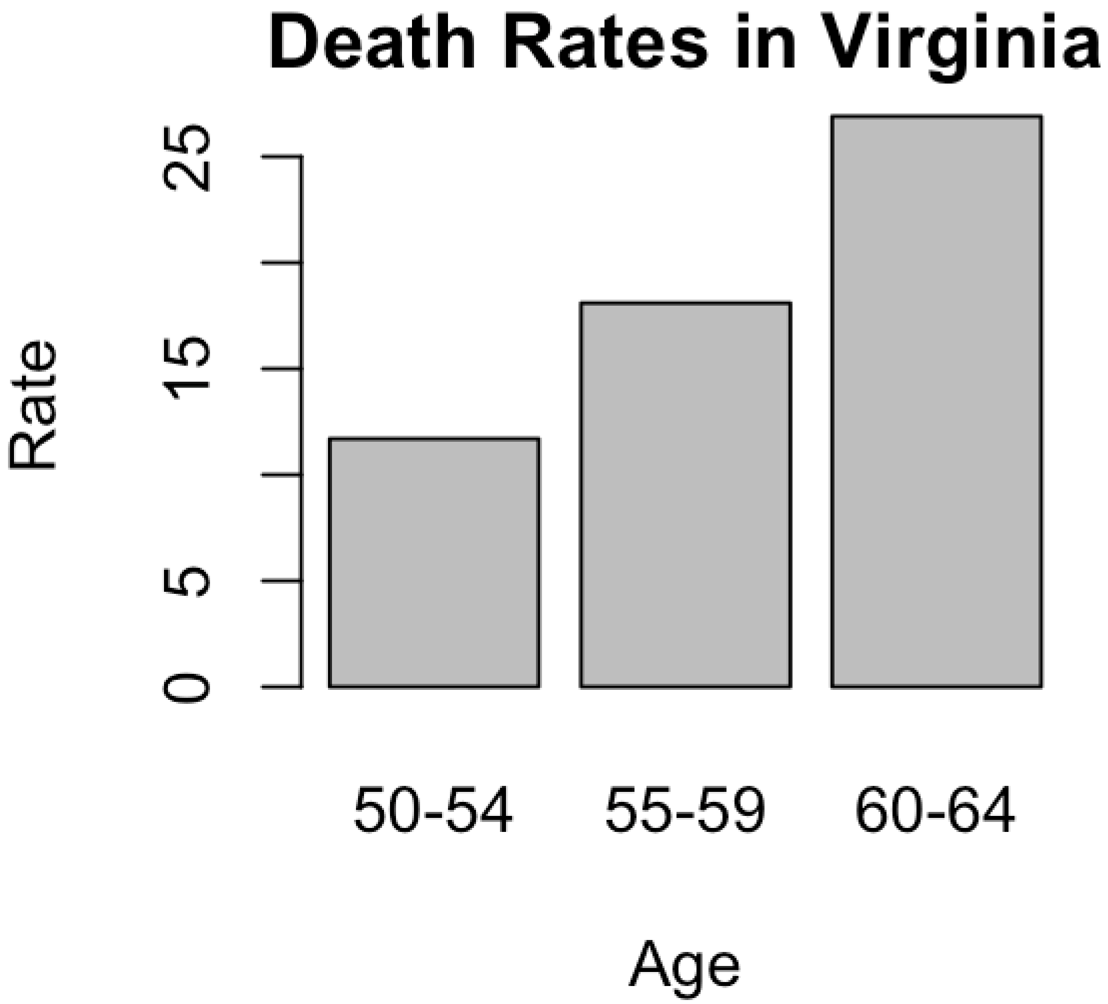
# Bar Plots

**Change main title and axis labels**

# Change axis titles

# Change color (col = "gray") and remove frame

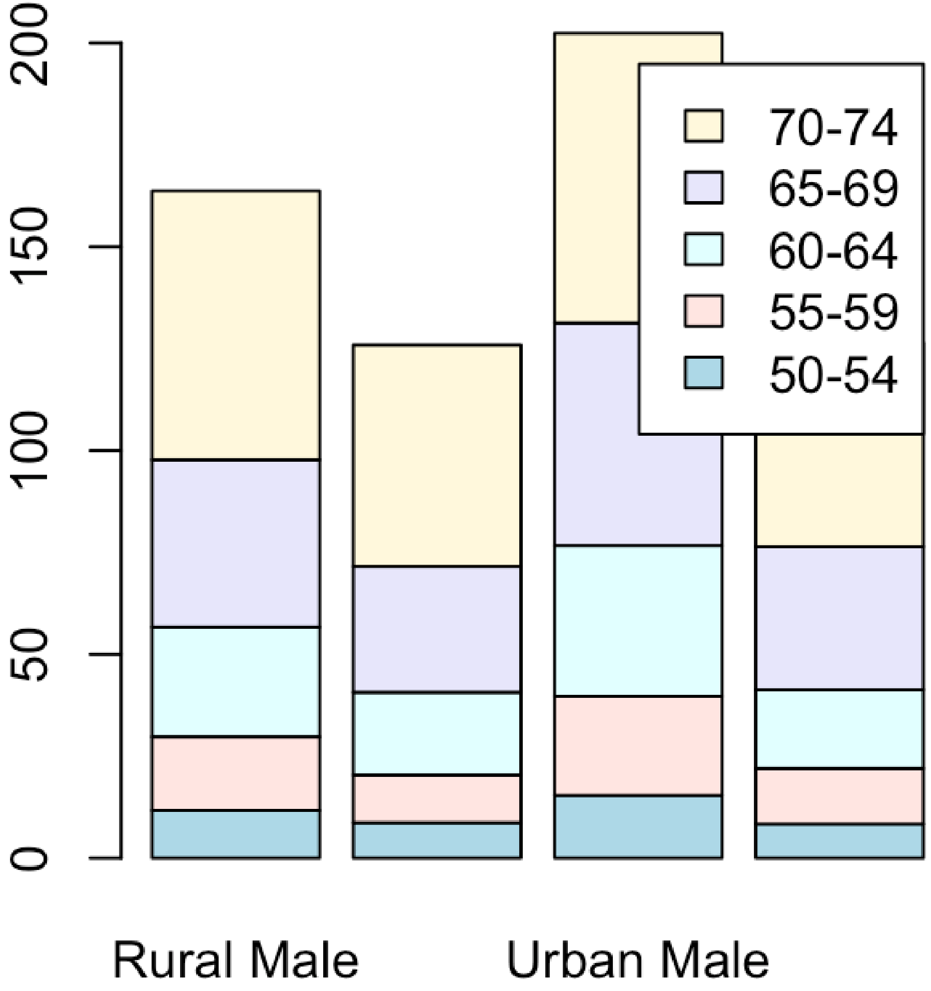
```
barplot(x, main = "Death Rates in Virginia",  xlab = "Age", ylab = "Rate")
```



# Bar Plots

## Stacked bar plots

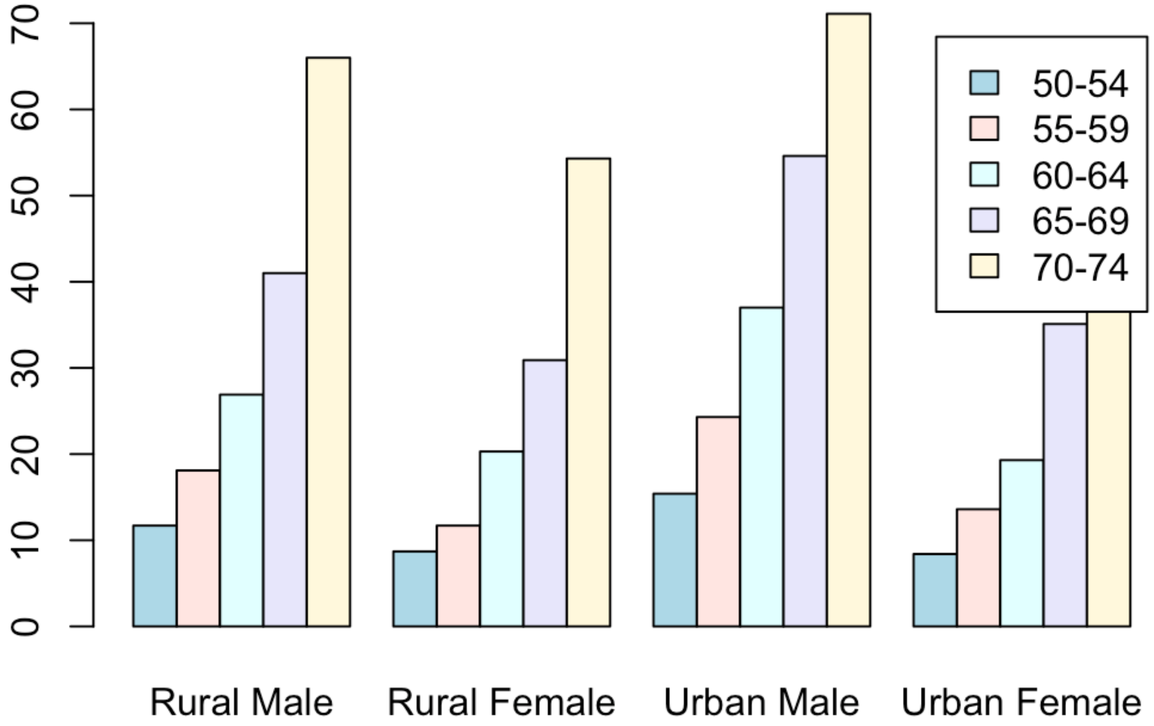
```
barplot(VADeaths,   col = c("lightblue",
"mistyrose", "lightcyan",  "lavender",
"cornsilk"),        legend =
rownames(VADeaths))
```



# Bar Plots

## Grouped bar plots

```
barplot(VADeaths,      col =
c("lightblue", "mistyrose", "lightcyan",

"lavender", "cornsilk"),   legend =
rownames(VADeaths), beside = TRUE)
```



# Bar Plots

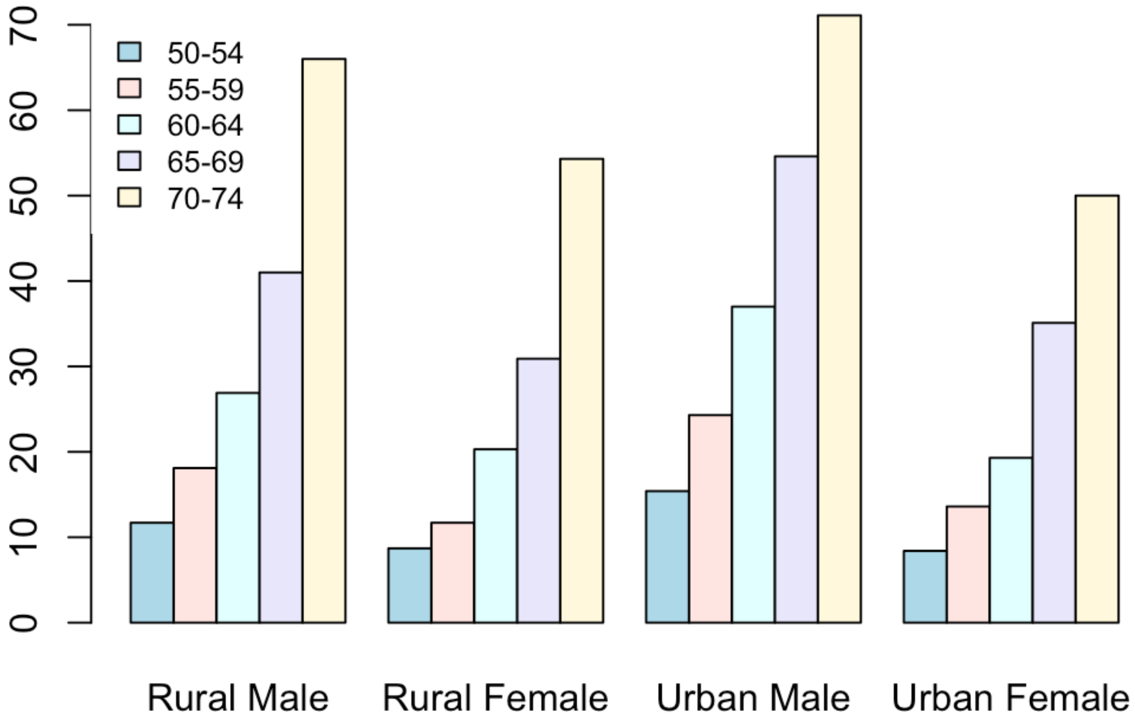
It’s also possible to add legends to a plot using the function legend() as follow.

```
# Define a set of colors
my_colors <- c("lightblue", "mistyrose", "lightcyan",
               "lavender", "cornsilk")
# Bar plot
barplot(VADeaths, col = my_colors, beside = TRUE)
# Add legend
legend("topleft", legend = rownames(VADeaths),
      fill = my_colors, box.lty = 0, cex = 0.8)
```

Notes:

**box.lty = 0**: Remove the box around the legend

**cex = 0.8**: legend text size



# Line Plots: plot() and lines()

The simplified format of plot() and lines() is as follow.

plot(x, y, type = "l", lty = 1)

lines(x, y, type = "l", lty = 1)

**x, y**: coordinate vectors of points to join

**type**: character indicating the type of plotting. Allowed values are:

- “p” for points
- “l” for lines
- “b” for both points and lines
- “c” for empty points joined by lines
- “o” for overplotted points and lines
- “s” and “S” for stair steps
- “n” does not produce any points or lines

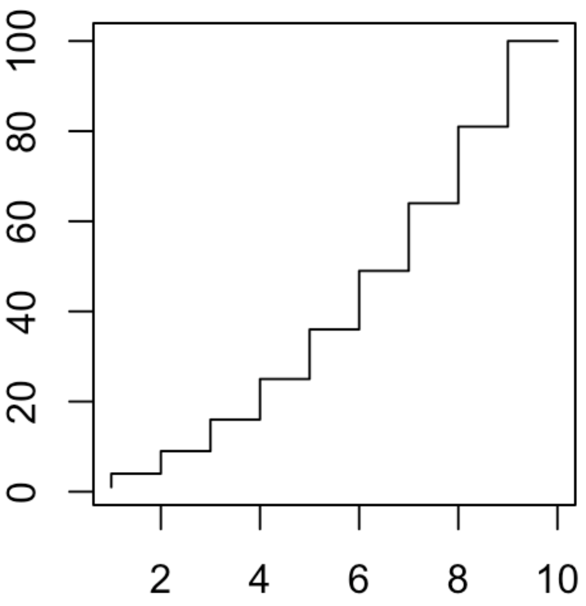
**lty**: line types.

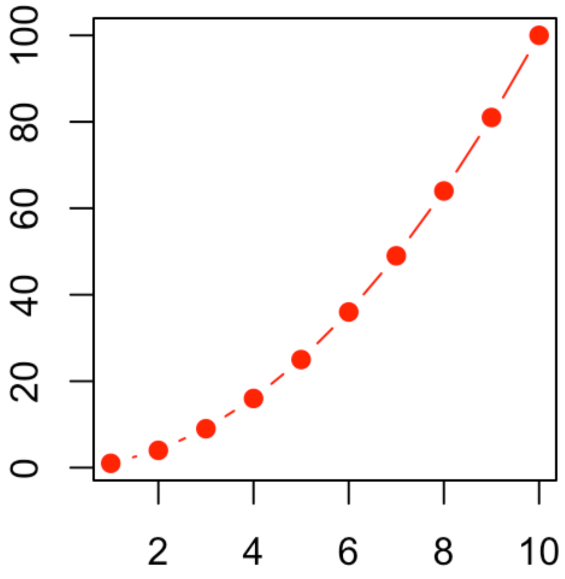
Line types can **either** be specified as **an integer** (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash)

or as one of the **character strings** “blank”, “solid”, “dashed”, “dotted”, “dotdash”, “longdash”, or “twodash”, where “blank” uses ‘invisible lines’ (i.e., does not draw them).

# Line Plots: plot() and lines()

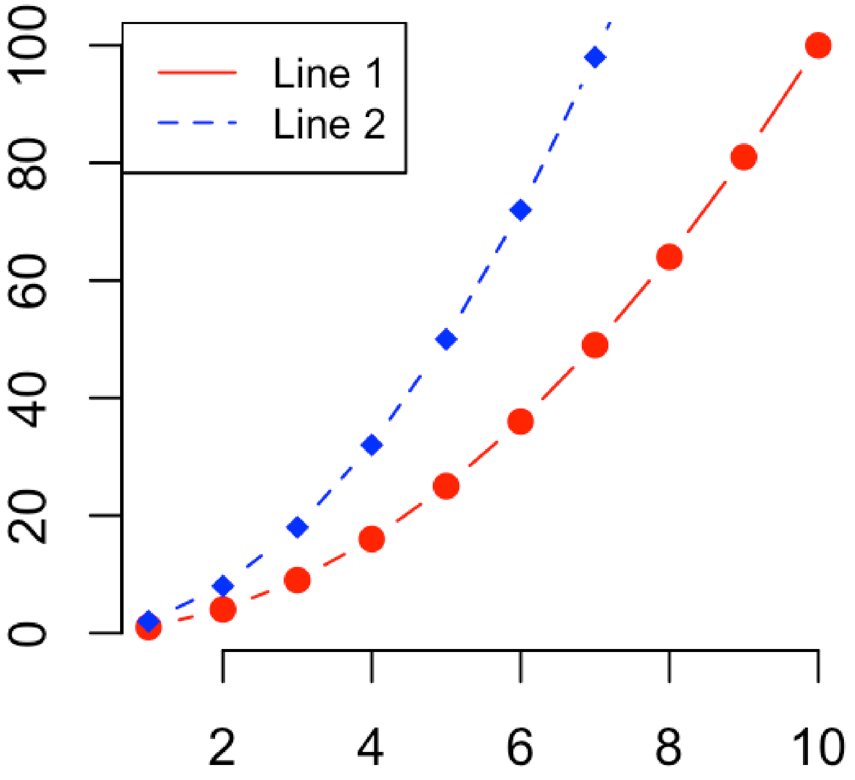
```
# Create some variables

x <- 1:10

y1 <- x*x

y2  <- 2*y1

# Create a basic stair steps plot

plot(x, y1, type = "S")

# Show both points and line

plot(x, y1, type = "b", pch = 19, col = "red",
xlab = "x", ylab = "y")
```





# Line Plots: plot() and lines()

```
# Create a first line

plot(x, y1, type = "b", frame = FALSE, pch = 19,
col = "red", xlab = "x", ylab = "y")

# Add a second line

lines(x, y2, pch = 18, col = "blue", type = "b",
lty = 2)

# Add a legend to the plot

legend("topleft", legend=c("Line 1", "Line 2"),
col=c("red", "blue"), lty = 1:2, cex=0.8)
```



# Pie Charts

**Pie charts are not recommended:** their features are somewhat limited.

Bar or dot plots recommended over pie charts because people are able to judge **length** more accurately than **volume**.

Pie charts are created with the function **pie(*x*, labels=)**

***x*** is a non-negative numeric vector indicating the area of each slice

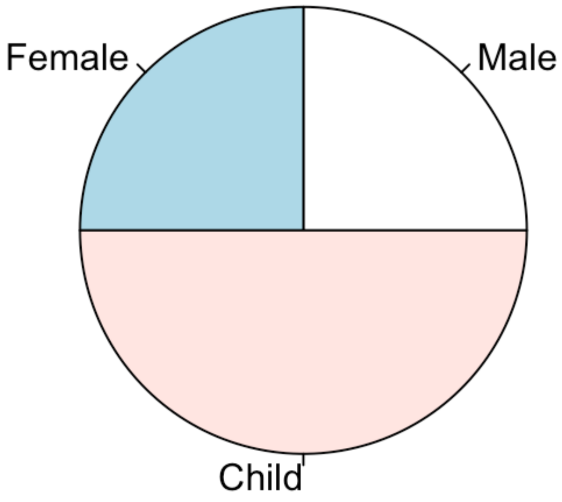
**labels**= notes a character vector of names for the slices.

**radius**: radius of the pie circle. If the character strings labeling the slices are long it may be necessary to use a smaller radius.

# Pie Charts: basic pie chart

Create some data

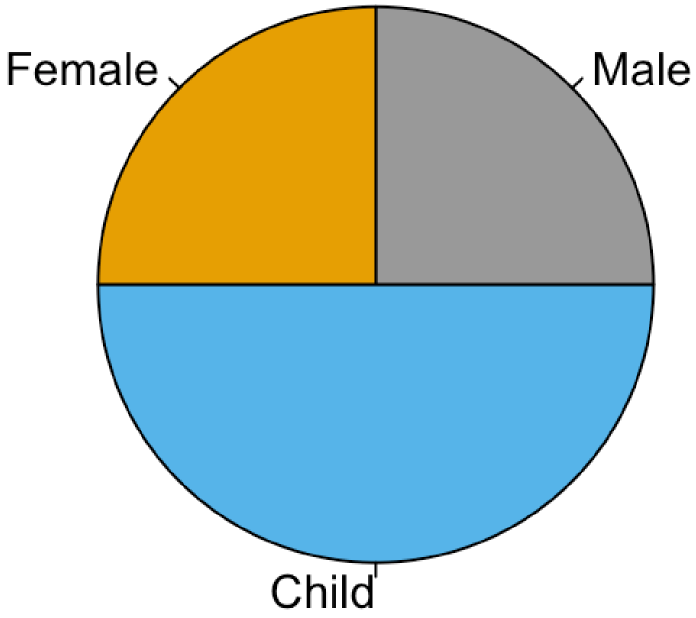
```
df <- data.frame(
  group = c("Male", "Female", "Child"),
  value = c(25, 25, 50)
  )
df


pie(df$value, labels = df$group, radius = 1)
```



# Pie Charts

# Change colors

```
pie(df$value, labels = df$group, radius = 1, col
= c("#999999", "#E69F00", "#56B4E9"))
```



# Pie Charts: Create 3D pie charts: plotix::pie3D()

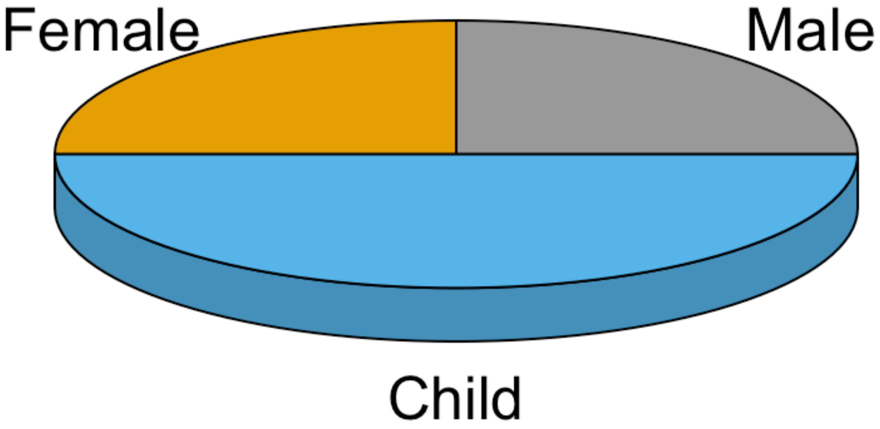
The function **pie3D**()[in **plotrix** package] can be used to draw a 3D pie chart.

**Install** plotrix package:

```
install.packages("plotrix")
```

Use **pie3D():**

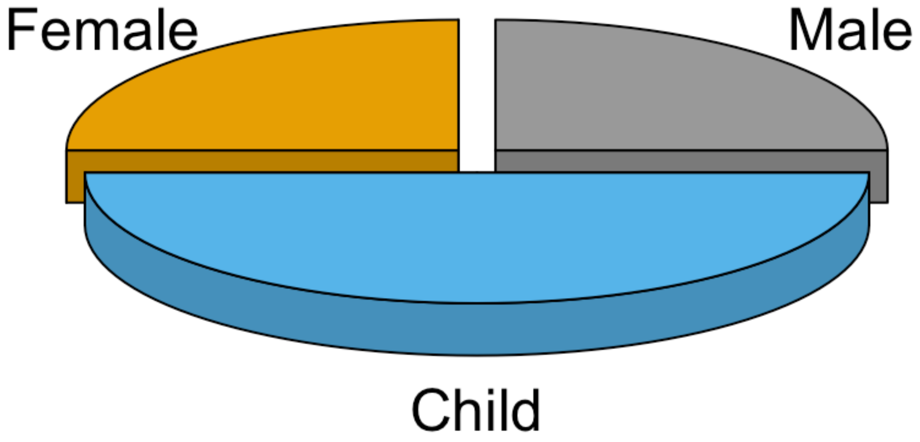
```
# 3D pie chart

library("plotrix")

pie3D(df$value, labels = df$group, radius = 1.5,  col = c("#999999", "#E69F00", "#56B4E9"))
```



# Pie Charts: Create 3D pie charts: plotix::pie3D()

# 3D Exploded Pie Chart

```
pie3D(df$value, labels = df$group, radius = 1.5,
    col = c("#999999", "#E69F00", "#56B4E9"),
    explode = 0.1)
```



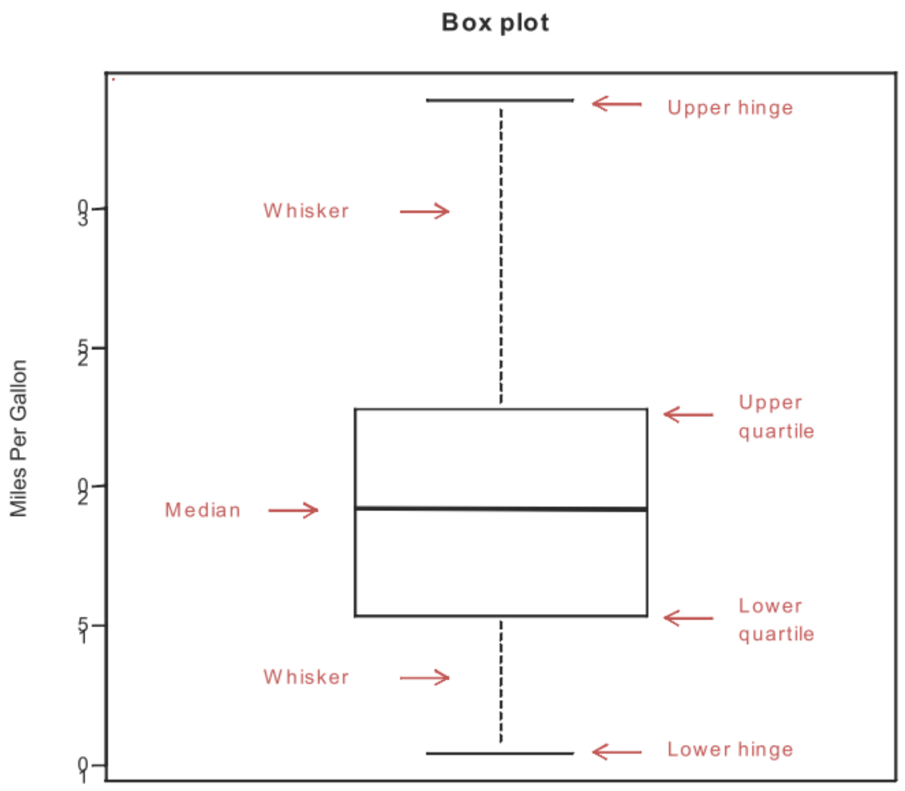
# Boxplots

A “box-and-whiskers” plot describes the distribution of a continuous variable by plotting its five-number summary: the **minimum**, **lower quartile** (25th percentile), **median** (50th percentile), **upper quartile** (75th percentile), and **maximum**.

It can also display observations that may be **outliers** (values outside the range of ± 1.5***IQR**,

where

IQR is the **interquartile range** defined as the **upper quartile minus the lower quartile**).



Box plot with annotations added by hand

# Boxplots

Boxplots can be created for **individual variables** or for **variables by group**.

The format is:

boxplot(x, data=),

where **x** is a formula

**data=** denotes the data frame providing the data.

Example formula is y~group - a separate boxplot for numeric variable **y** is generated for each value of **group**.

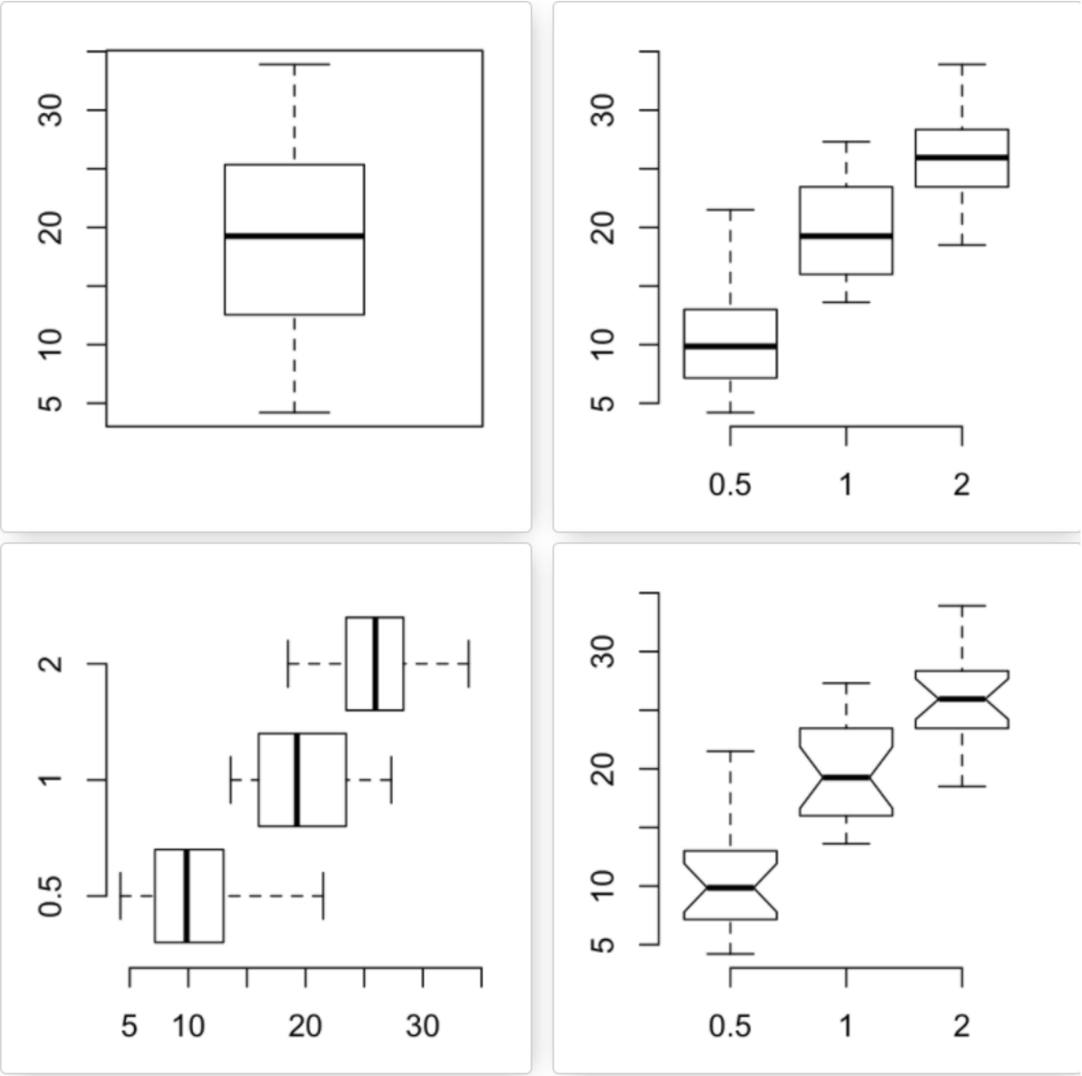
**varwidth=TRUE:** boxplot widths are made proportional to the square root of the samples sizes.

**horizontal=TRUE** reverses the axis orientation.

# Boxplots

Here, we'll use the R built-in ToothGrowth data set.

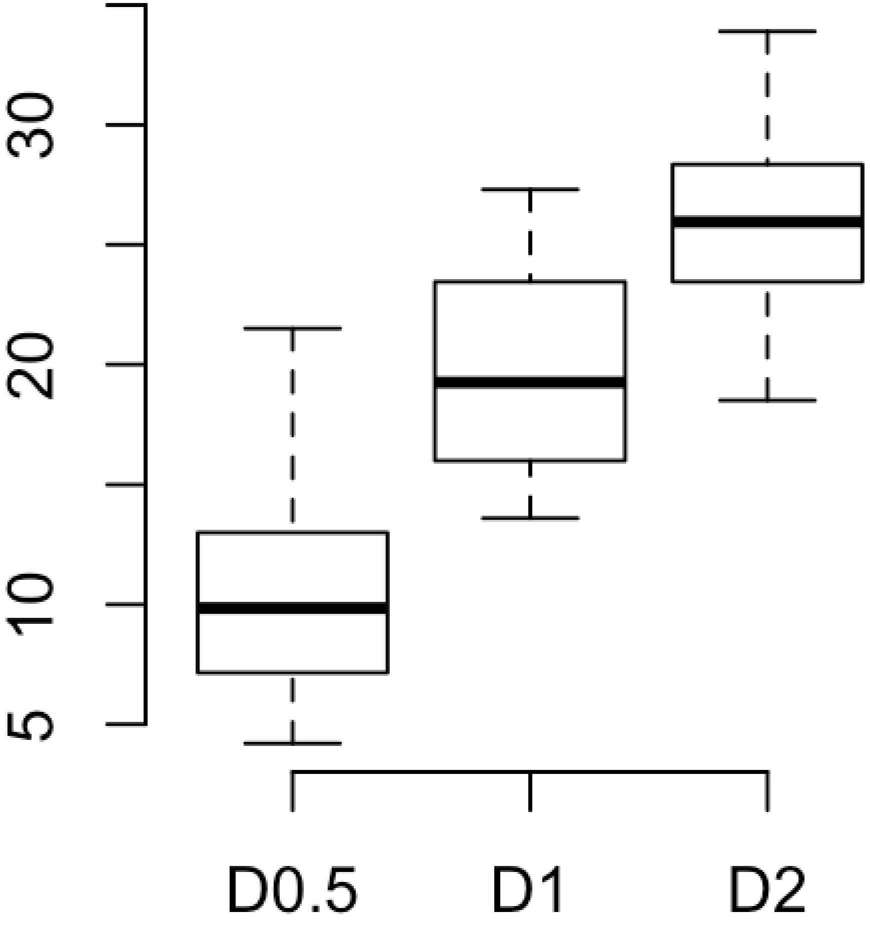
```
# Box plot of one variable
boxplot(ToothGrowth$len)
# Box plots by groups (dose) removing frame
boxplot(len ~ dose, data = ToothGrowth, frame = FALSE)
# Horizontal box plots
boxplot(len ~ dose, data = ToothGrowth, frame = FALSE,
    horizontal = TRUE)
# Notched box plots
boxplot(len ~ dose, data = ToothGrowth, frame = FALSE,
    notch = TRUE)
```



# Boxplots

**Change group names**

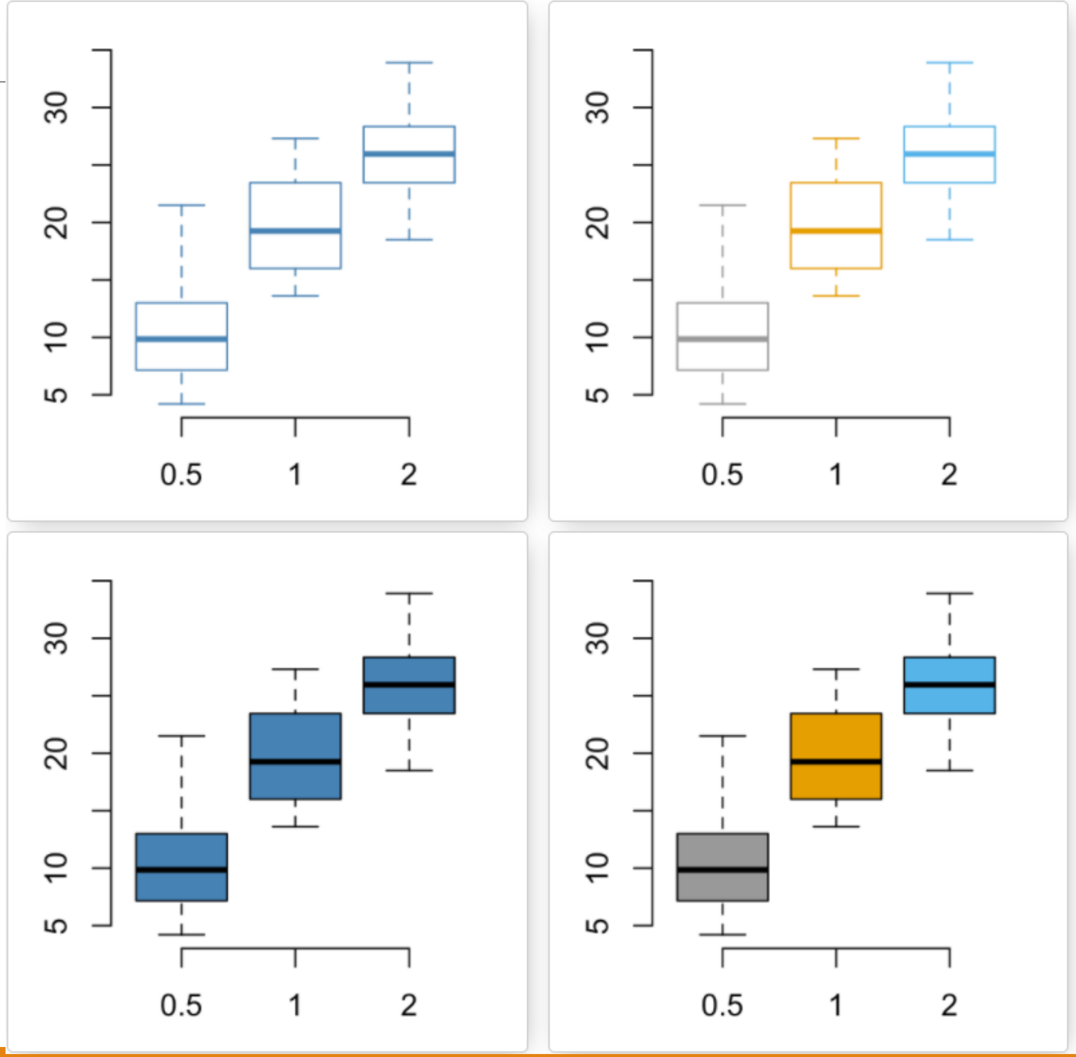
```
boxplot(len ~ dose, data = ToothGrowth, frame
= FALSE, names = c("D0.5", "D1", "D2"))
```



# Boxplots

**Change color**

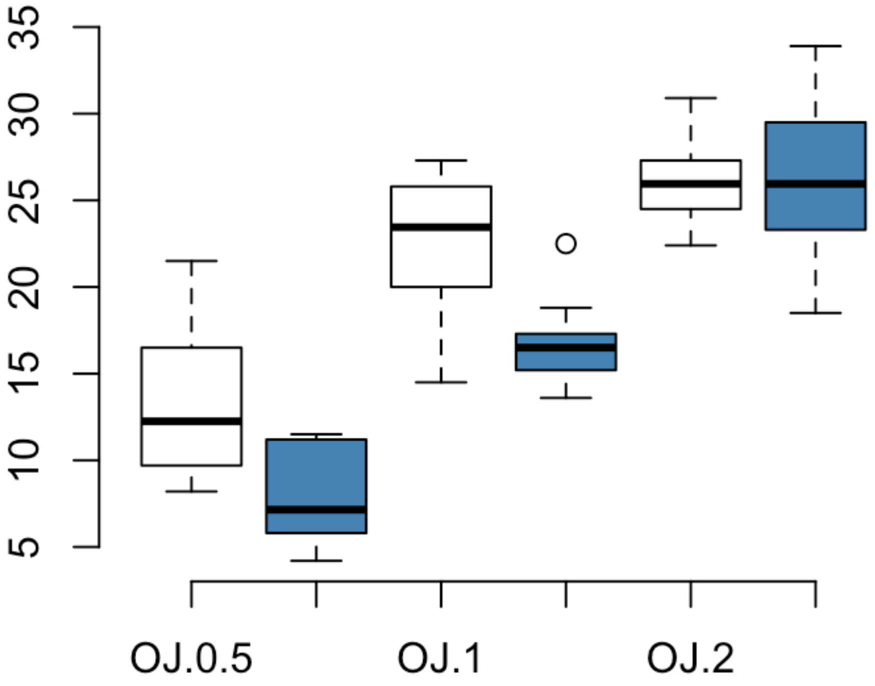
```
# Change the color of border using one single color
boxplot(len ~ dose, data = ToothGrowth, frame = FALSE,
        border = "steelblue")
#  Use different border colors for each group
boxplot(len ~ dose, data = ToothGrowth, frame = FALSE,
        border = c("#999999", "#E69F00", "#56B4E9"))
# Change fill color : single color
boxplot(len ~ dose, data = ToothGrowth, frame = FALSE,
        col = "steelblue")
# Change fill color: multiple colors
boxplot(len ~ dose, data = ToothGrowth, frame = FALSE,
        col = c("#999999", "#E69F00", "#56B4E9"))
```



# Boxplots

Box plot with multiple groups

```
boxplot(len ~ supp*dose, data = ToothGrowth,
        col = c("white", "steelblue"), frame =
FALSE)
```



# Boxplots

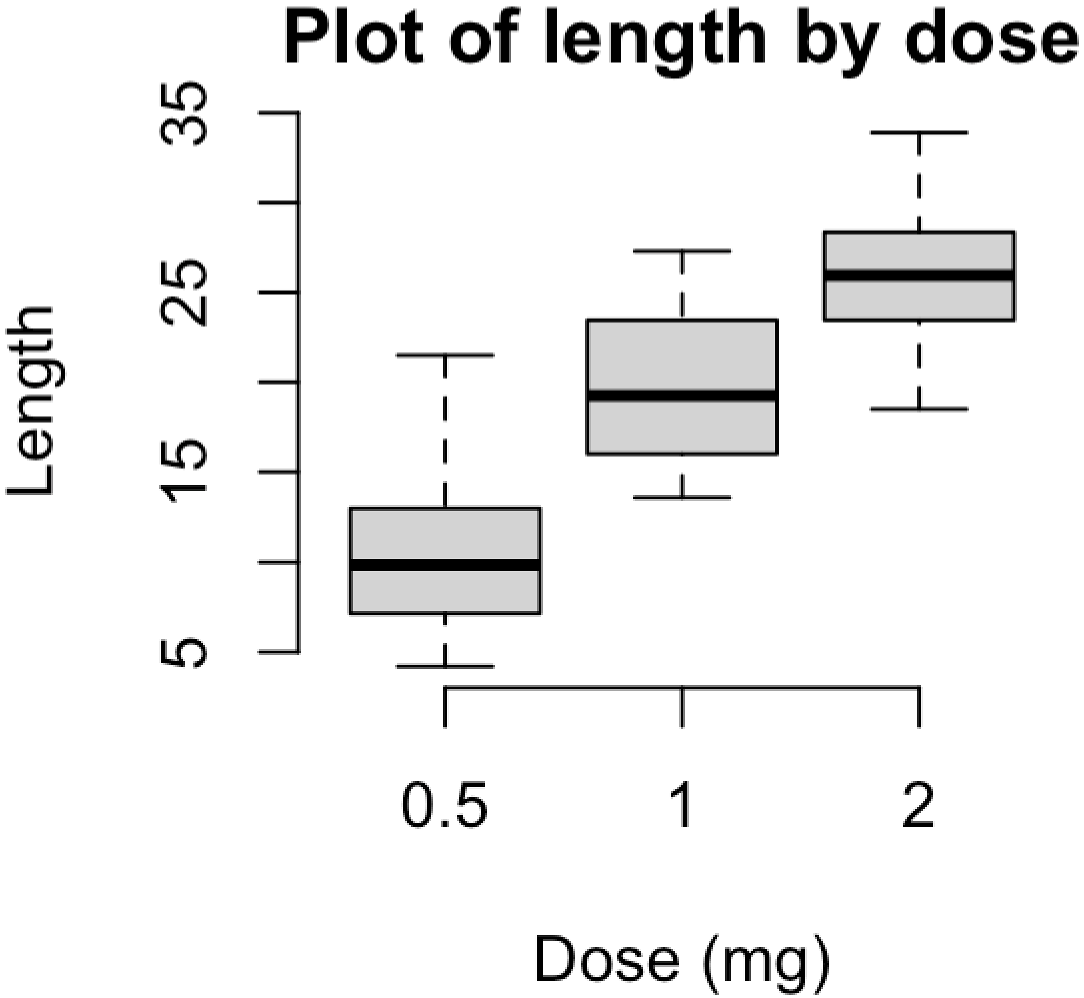
Change main title and axis labels

# Change axis titles

# Change color (col = "gray") and remove frame

# Create notched box plot

```
boxplot(len ~ dose, data = ToothGrowth,
    main = "Plot of length by dose",
    xlab = "Dose (mg)", ylab = "Length",
    col = "lightgray", frame = FALSE)
```



# Scatterplots

A convenient method of plotting a bivariate relationship- relationships between two variables (more statistical details to be covered in a later session)

Scatter plots can be created using the function plot(x, y).

The function lm() will be used to fit linear models between y and x.

A regression line will be added on the plot using the function abline(), which takes the output of lm() as an argument.

Smoothing lines can be added using the function loess().

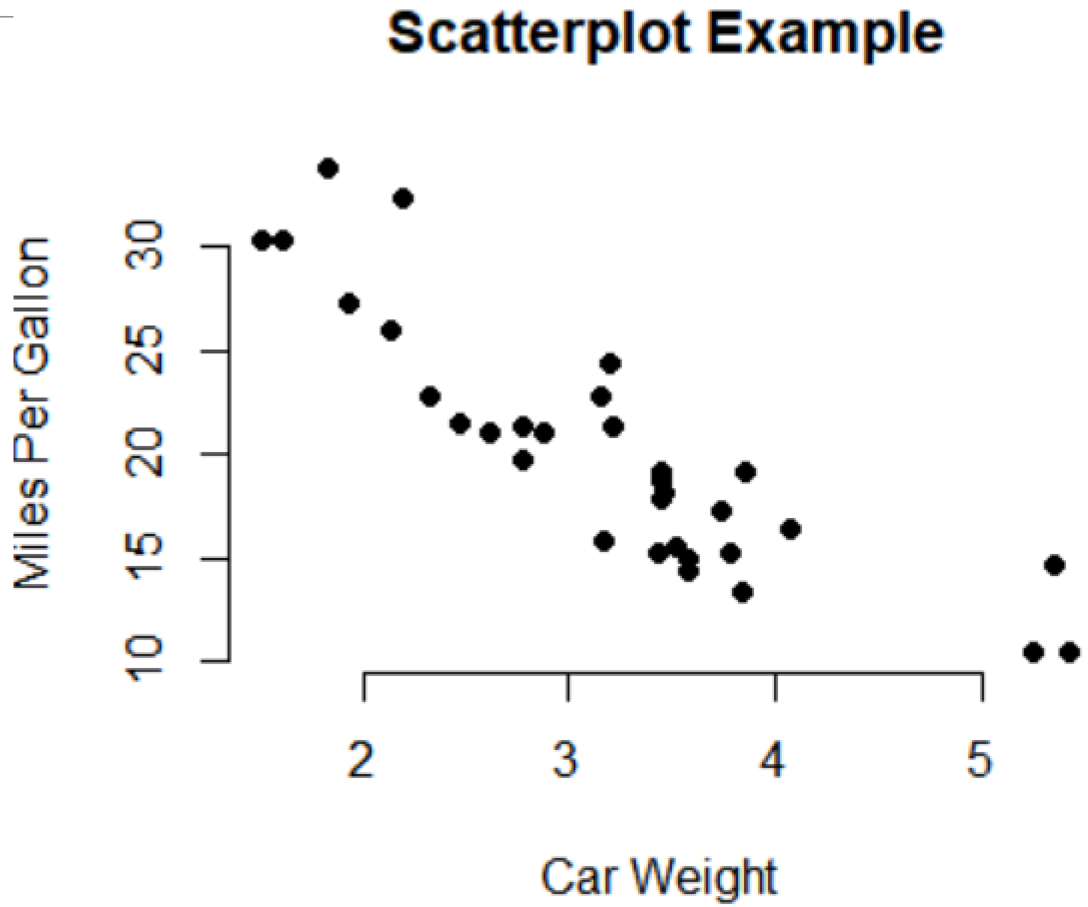
# Scatterplots: basic

```
attach(mtcars)
```

# Plot with main and axis titles

# Change point shape (pch = 19) and remove frame.

```
plot(wt, mpg, main="Scatterplot Example",
  xlab="Car Weight ", ylab="Miles Per Gallon ",
pch=19, frame=FALSE)
```



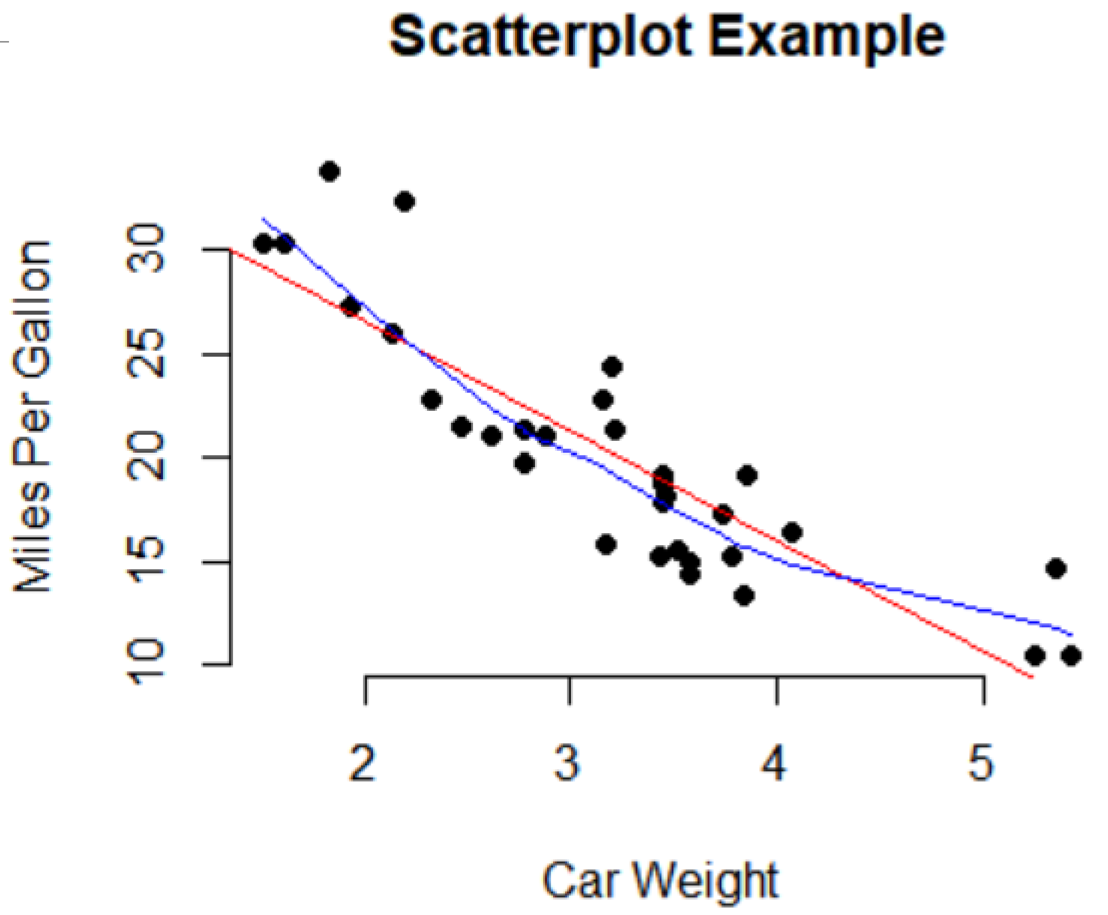
# Scatterplots

# Add regression line

```
plot(wt, mpg, main="Scatterplot Example",
  xlab="Car Weight ", ylab="Miles Per Gallon ",
pch=19, frame=FALSE)
```

# Add fit lines

```
abline(lm(mpg~wt), col="red") # regression line (y~x)
lines(lowess(wt,mpg), col="blue") # lowess line (x,y)
```



# Enhanced scatter plots: car::scatterplot()

The function scatterplot() [in car package] makes enhanced scatter plots, with box plots in the margins, a non-parametric regression smooth, smoothed conditional spread, outlier identification, and a regression line, …

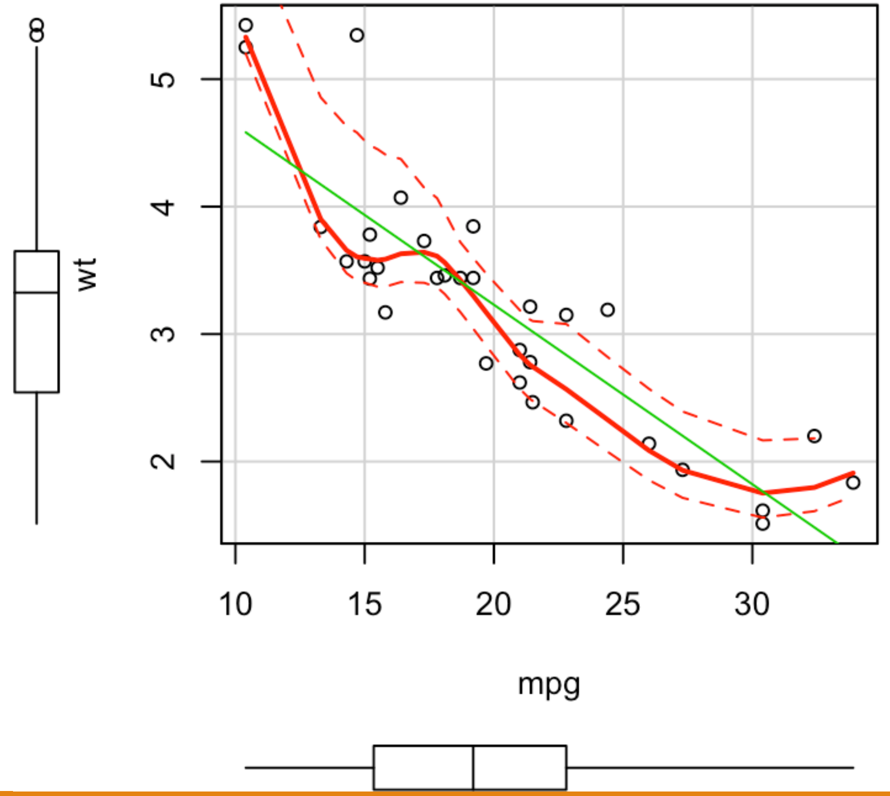
Install car package:

install.packages("car")

Use scatterplot() function:

library("car")

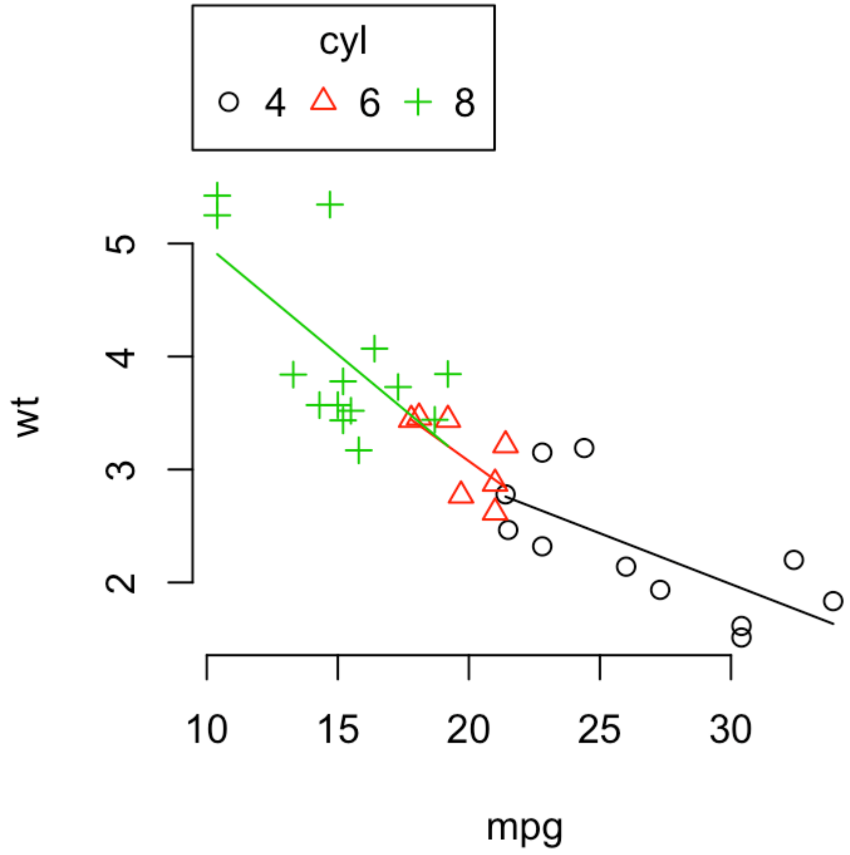
scatterplot(wt ~ mpg, data = mtcars)



# Scatterplots: grouped

# Scatter plot by groups ("cyl")

```
scatterplot(wt ~ mpg | cyl, data = mtcars,
smoother = FALSE, grid = FALSE, frame =
FALSE)
```

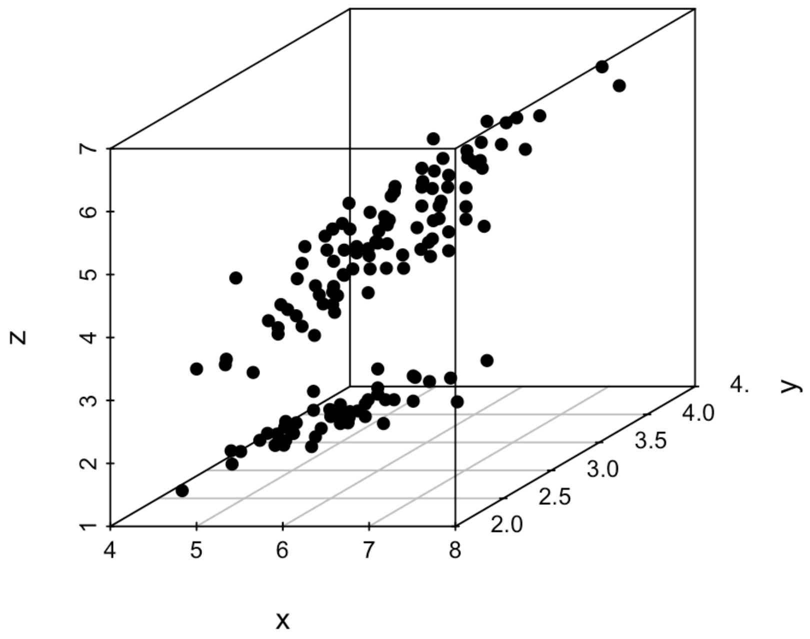


# Scatterplots: 3D

Function scatterplot3D [in scatterplot3D package can be used].

The following R code plots a 3D scatter plot using iris data set.

```
# Prepare the data set

x <- iris$Sepal.Length

y <- iris$Sepal.Width

z <- iris$Petal.Length

grps <- as.factor(iris$Species)

# Plot

library(scatterplot3d)

scatterplot3d(x, y, z, pch = 16)
```



# Scatterplots: 3D

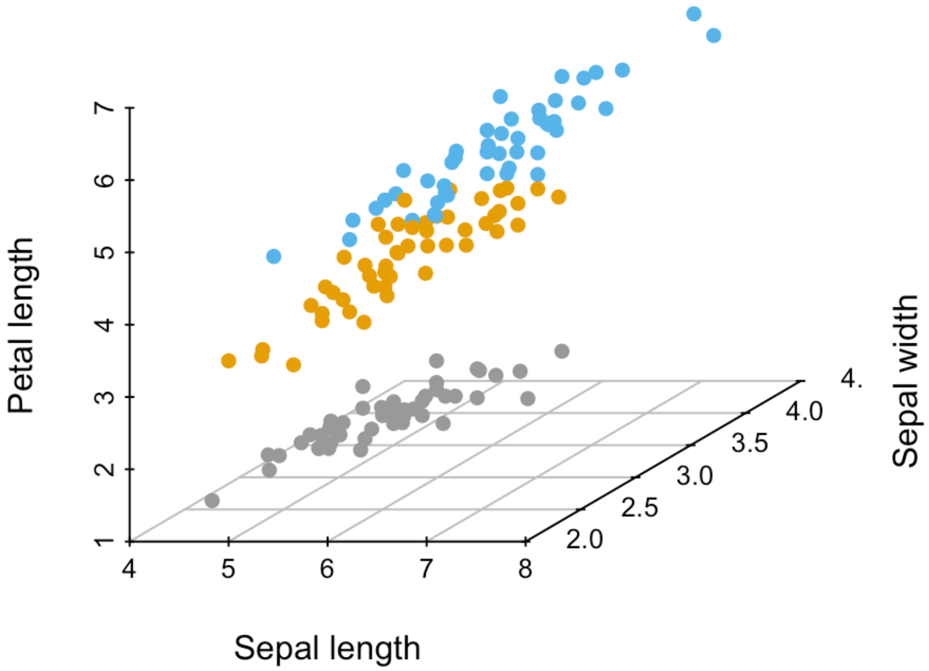
Change color by groups

Add grids and remove the box around the plot

Change axis labels: xlab, ylab and zlab

```
colors <- c("#999999", "#E69F00", "#56B4E9")
```

```
scatterplot3d(x, y, z, pch = 16, color = colors[grps],    grid = TRUE, box = FALSE, xlab = "Sepal length", ylab = "Sepal width", zlab = "Petal length")
```



# Advanced Graphics
# - Graphical parameters
# - Axes and text
# - Combining plots

# Graphical Parameters

- You can customize many features of your graphs (fonts, colors, axes, titles) through graphic options.
- The par( ) function: parameter values set here are in effect for the rest of the session or until you change them again.
- The format is par(*optionname*=*value*, *optionname*=*value*, ...)
- # Set a graphical parameter using par()

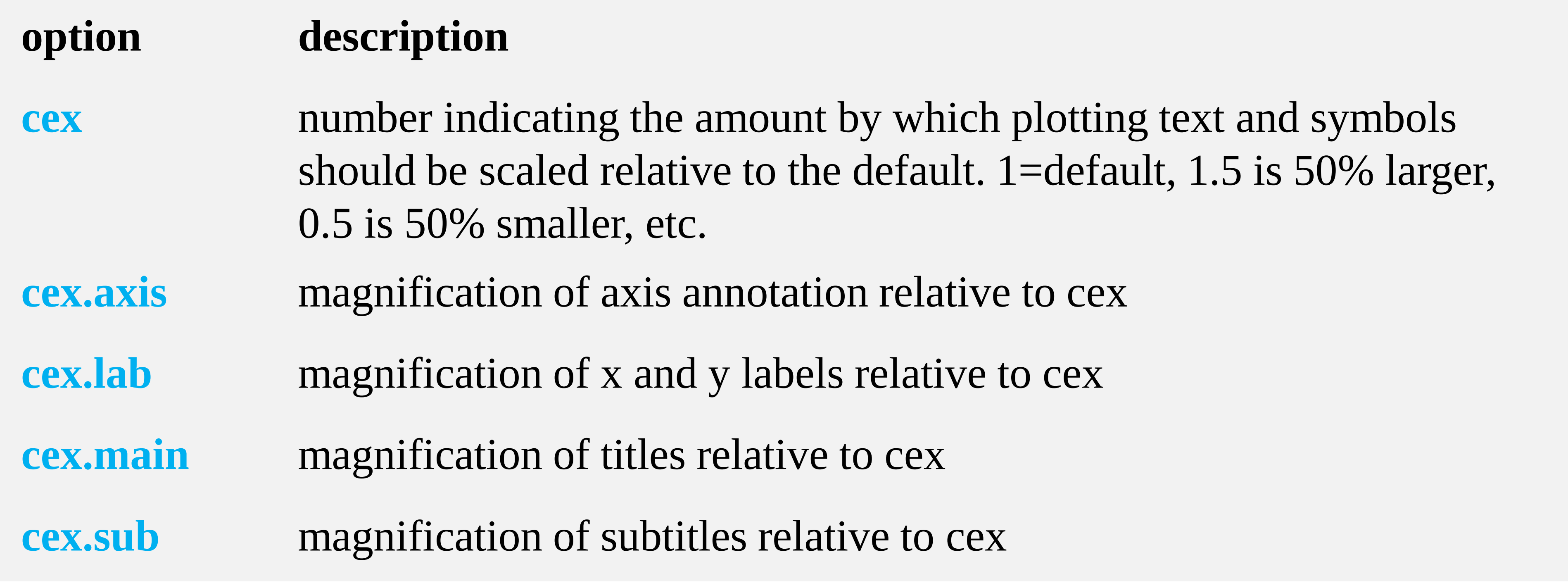
```
par()              # view current settings
opar <- par()      # make a copy of current settings
par(col.lab="red") # red x and y labels
hist(mtcars$mpg)   # create a plot with these new settings
par(opar)          # restore original settings
```

# Graphical Parameters

- A second way to specify graphical parameters is by providing the *optionname=value* pairs directly to a high-level plotting function. In this case, the options are only in effect for that specific graph.
- # Set a graphical parameter within the plotting function
  `hist(mtcars$mpg, col.lab="red")`
- See the help for a specific high level plotting function (e.g. plot, hist, boxplot) to determine which graphical parameters can be set this way.
- The remainder of this section describes some of the more important graphical parameters that you can set.

# Graphical Parameters

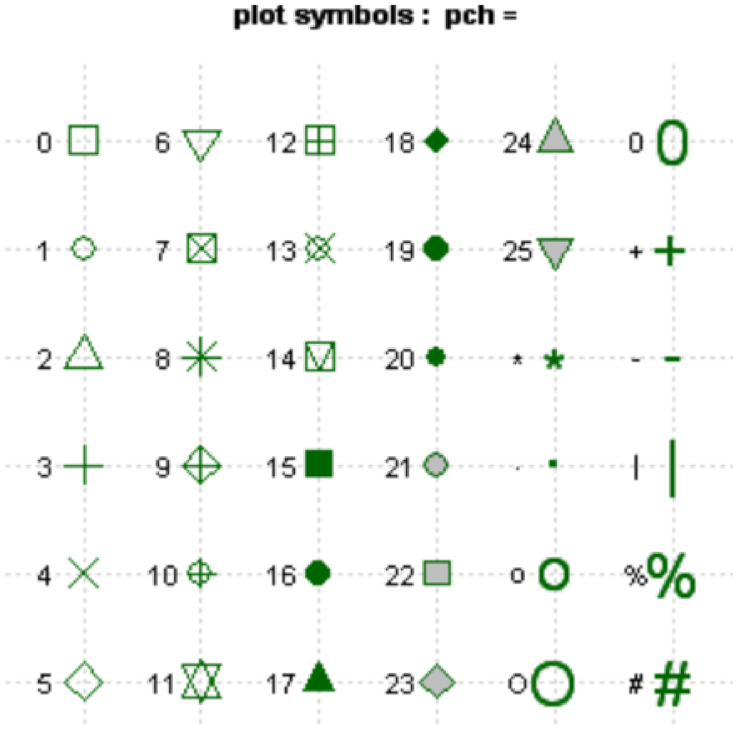
- **Text and Symbol Size**
- The following options can be used to control text and symbol size in graphs.



| option | description |
|---|---|
| cex | number indicating the amount by which plotting text and symbols should be scaled relative to the default. 1=default, 1.5 is 50% larger, 0.5 is 50% smaller, etc. |
| cex.axis | magnification of axis annotation relative to cex |
| cex.lab | magnification of x and y labels relative to cex |
| cex.main | magnification of titles relative to cex |
| cex.sub | magnification of subtitles relative to cex |

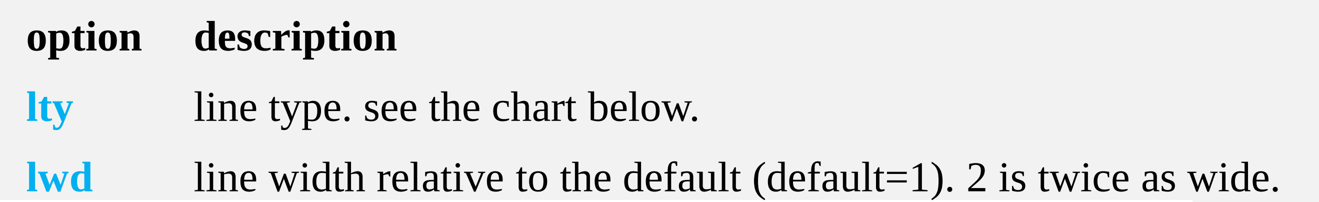
# Graphical Parameters

- **PLOTTING SYMBOLS**
- Use the **pch=** option to specify symbols to use when plotting points. For symbols 21 through 25, specify border color (col=) and fill color (bg=).

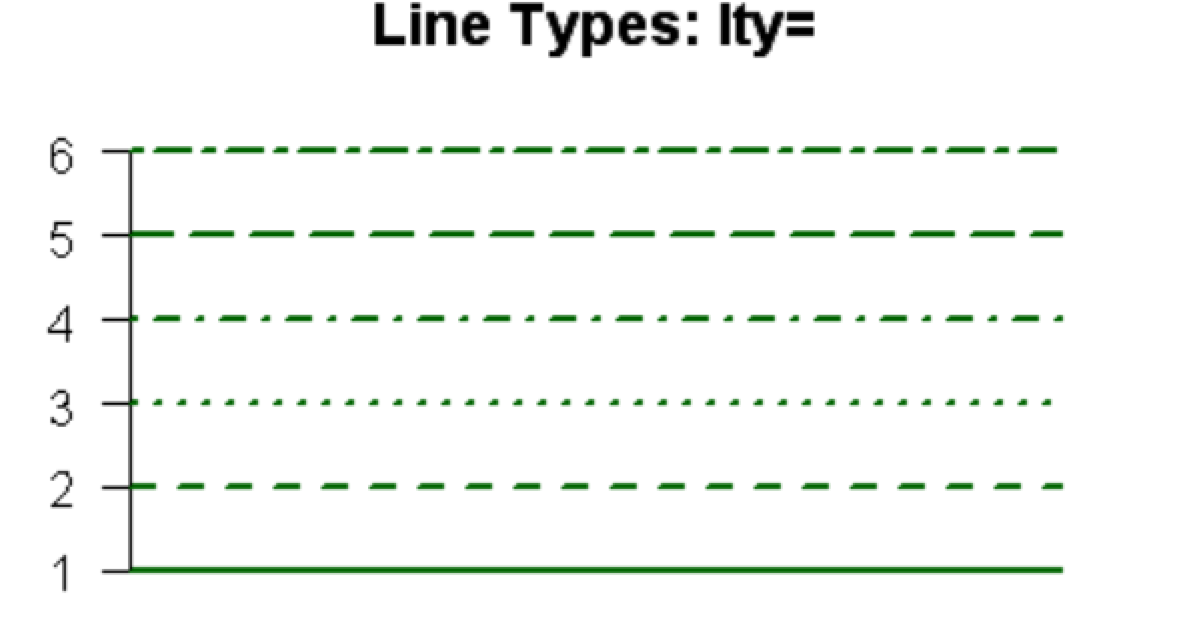


# Graphical Parameters

- LINES
- You can change lines using the following options. This is particularly useful for reference lines, axes, and fit lines.



| option | description |
|---|---|
| lty | line type. see the chart below. |
| lwd | line width relative to the default (default=1). 2 is twice as wide. |



# Graphical Parameters

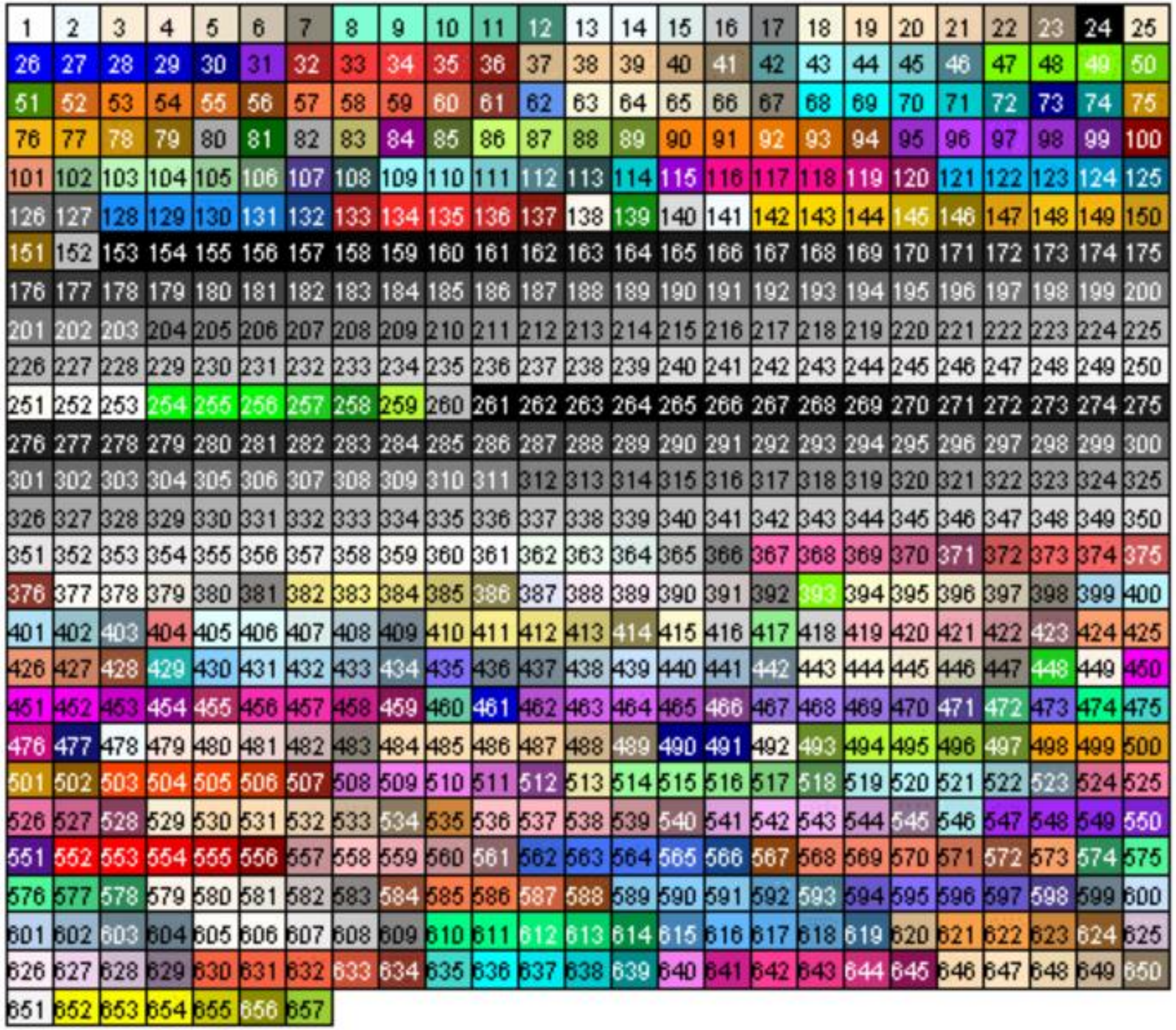
- **COLORS**
- Options that specify colors include the following.

| **option** | **description** |
|---|---|
| **col** | Default plotting color. Some functions (e.g. lines) accept a vector of values that are recycled. |
| **col.axis** | color for axis annotation |
| **col.lab** | color for x and y labels |
| **col.main** | color for titles |
| **col.sub** | color for subtitles |
| **fg** | plot foreground color (axes, boxes - also sets col= to same) |
| **bg** | plot background color |

# Graphical Parameters

- You can specify colors in R by index, name, hexadecimal, or RGB.
  For example col=1, col="white", and col="#FFFFFF" are equivalent.
- The following slide prersents a chart was produced with code developed by Earl F. Glynn.
- You can also create a vector of *n* contiguous colors using the functions **rainbow(*n*)**, **heat.colors(*n*)**, **terrain.colors(*n*)**, **topo.colors(*n*)**, and **cm.colors(*n*)**.
- **colors()** returns all available color names.

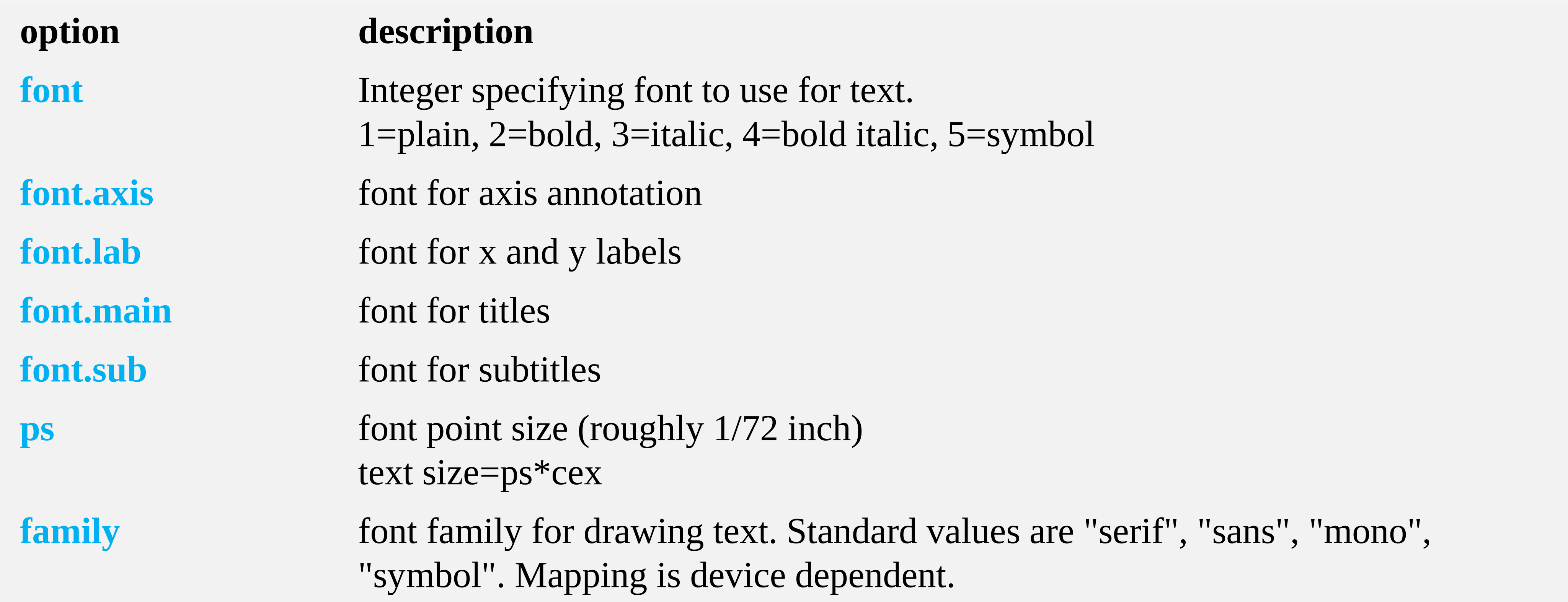
# Graphical Parameters: color code chart



| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 |
| 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |
| 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 |
| 126 | 127 | 128 | 129 | 130 | 131 | 132 | 133 | 134 | 135 | 136 | 137 | 138 | 139 | 140 | 141 | 142 | 143 | 144 | 145 | 146 | 147 | 148 | 149 | 150 |
| 151 | 152 | 153 | 154 | 155 | 156 | 157 | 158 | 159 | 160 | 161 | 162 | 163 | 164 | 165 | 166 | 167 | 168 | 169 | 170 | 171 | 172 | 173 | 174 | 175 |
| 176 | 177 | 178 | 179 | 180 | 181 | 182 | 183 | 184 | 185 | 186 | 187 | 188 | 189 | 190 | 191 | 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 200 |
| 201 | 202 | 203 | 204 | 205 | 206 | 207 | 208 | 209 | 210 | 211 | 212 | 213 | 214 | 215 | 216 | 217 | 218 | 219 | 220 | 221 | 222 | 223 | 224 | 225 |
| 226 | 227 | 228 | 229 | 230 | 231 | 232 | 233 | 234 | 235 | 236 | 237 | 238 | 239 | 240 | 241 | 242 | 243 | 244 | 245 | 246 | 247 | 248 | 249 | 250 |
| 251 | 252 | 253 | 254 | 255 | 256 | 257 | 258 | 259 | 260 | 261 | 262 | 263 | 264 | 265 | 266 | 267 | 268 | 269 | 270 | 271 | 272 | 273 | 274 | 275 |
| 276 | 277 | 278 | 279 | 280 | 281 | 282 | 283 | 284 | 285 | 286 | 287 | 288 | 289 | 290 | 291 | 292 | 293 | 294 | 295 | 296 | 297 | 298 | 299 | 300 |
| 301 | 302 | 303 | 304 | 305 | 306 | 307 | 308 | 309 | 310 | 311 | 312 | 313 | 314 | 315 | 316 | 317 | 318 | 319 | 320 | 321 | 322 | 323 | 324 | 325 |
| 326 | 327 | 328 | 329 | 330 | 331 | 332 | 333 | 334 | 335 | 336 | 337 | 338 | 339 | 340 | 341 | 342 | 343 | 344 | 345 | 346 | 347 | 348 | 349 | 350 |
| 351 | 352 | 353 | 354 | 355 | 356 | 357 | 358 | 359 | 360 | 361 | 362 | 363 | 364 | 365 | 366 | 367 | 368 | 369 | 370 | 371 | 372 | 373 | 374 | 375 |
| 376 | 377 | 378 | 379 | 380 | 381 | 382 | 383 | 384 | 385 | 386 | 387 | 388 | 389 | 390 | 391 | 392 | 393 | 394 | 395 | 396 | 397 | 398 | 399 | 400 |
| 401 | 402 | 403 | 404 | 405 | 406 | 407 | 408 | 409 | 410 | 411 | 412 | 413 | 414 | 415 | 416 | 417 | 418 | 419 | 420 | 421 | 422 | 423 | 424 | 425 |
| 426 | 427 | 428 | 429 | 430 | 431 | 432 | 433 | 434 | 435 | 436 | 437 | 438 | 439 | 440 | 441 | 442 | 443 | 444 | 445 | 446 | 447 | 448 | 449 | 450 |
| 451 | 452 | 453 | 454 | 455 | 456 | 457 | 458 | 459 | 460 | 461 | 462 | 463 | 464 | 465 | 466 | 467 | 468 | 469 | 470 | 471 | 472 | 473 | 474 | 475 |
| 476 | 477 | 478 | 479 | 480 | 481 | 482 | 483 | 484 | 485 | 486 | 487 | 488 | 489 | 490 | 491 | 492 | 493 | 494 | 495 | 496 | 497 | 498 | 499 | 500 |
| 501 | 502 | 503 | 504 | 505 | 506 | 507 | 508 | 509 | 510 | 511 | 512 | 513 | 514 | 515 | 516 | 517 | 518 | 519 | 520 | 521 | 522 | 523 | 524 | 525 |
| 526 | 527 | 528 | 529 | 530 | 531 | 532 | 533 | 534 | 535 | 536 | 537 | 538 | 539 | 540 | 541 | 542 | 543 | 544 | 545 | 546 | 547 | 548 | 549 | 550 |
| 551 | 552 | 553 | 554 | 555 | 556 | 557 | 558 | 559 | 560 | 561 | 562 | 563 | 564 | 565 | 566 | 567 | 568 | 569 | 570 | 571 | 572 | 573 | 574 | 575 |
| 576 | 577 | 578 | 579 | 580 | 581 | 582 | 583 | 584 | 585 | 586 | 587 | 588 | 589 | 590 | 591 | 592 | 593 | 594 | 595 | 596 | 597 | 598 | 599 | 600 |
| 601 | 602 | 603 | 604 | 605 | 606 | 607 | 608 | 609 | 610 | 611 | 612 | 613 | 614 | 615 | 616 | 617 | 618 | 619 | 620 | 621 | 622 | 623 | 624 | 625 |
| 626 | 627 | 628 | 629 | 630 | 631 | 632 | 633 | 634 | 635 | 636 | 637 | 638 | 639 | 640 | 641 | 642 | 643 | 644 | 645 | 646 | 647 | 648 | 649 | 650 |
| 651 | 652 | 653 | 654 | 655 | 656 | 657 | | | | | | | | | | | | | | | | | | |

# Graphical Parameters

- fonts
- You can easily set font size and style, but font family is a bit more complicated.



| option | description |
|---|---|
| font | Integer specifying font to use for text.<br>1=plain, 2=bold, 3=italic, 4=bold italic, 5=symbol |
| font.axis | font for axis annotation |
| font.lab | font for x and y labels |
| font.main | font for titles |
| font.sub | font for subtitles |
| ps | font point size (roughly 1/72 inch)<br>text size=ps*cex |
| family | font family for drawing text. Standard values are "serif", "sans", "mono", "symbol". Mapping is device dependent. |

# Graphical Parameters

- Axes and Text
- Many high-level plotting functions (plot, hist, boxplot, etc.) allow you to include axis and text options (as well as other graphical paramters). For example

-
  ```
  # Specify axis options within plot()
  plot(x, y, main="title", sub="subtitle",
    xlab="X-axis label", ylab="y-axix label",
    xlim=c(xmin, xmax), ylim=c(ymin, ymax))
  ```
- For finer control or for modularization, you can use the functions described below.

# Graphical Parameters

- Titles
- Use the title( ) function to add labels to a plot.
- title(main="*main title*", sub="*sub-title*",
  xlab="*x-axis label*", ylab="*y-axis label*")
- Many other ***graphical parameters*** (such as text size, font, rotation, and color) can also be specified in the title( ) function.
- # Add a red title and a blue subtitle. Make x and y
  # labels 25% smaller than the default and green.

```
# Add a red title and a blue subtitle. Make x and y
# labels 25% smaller than the default and green.
title(main="My Title", col.main="red",
  sub="My Sub-title", col.sub="blue",
  xlab="My X label", ylab="My Y label",
  col.lab="green", cex.lab=0.75)
```

# Graphical Parameters

If you are going to create a custom axis, you should suppress the axis automatically generated by your high level plotting function.

The option axes=FALSE suppresses both x and y axes. xaxt="n" and yaxt="n" suppress the x and y axis respectively.

# Graphical Parameters

## Axes

You can create custom axes using the **axis( )** function.

axis(*side*, at=, labels=, pos=, lty=, col=, las=, tck=, ...)

where

| **option** | **description** |
|---|---|
| **side** | an integer indicating the side of the graph to draw the axis (1=bottom, 2=left, 3=top, 4=right) |
| **at** | a numeric vector indicating where tic marks should be drawn |
| **labels** | a character vector of labels to be placed at the tickmarks<br>(if NULL, the *at* values will be used) |
| **pos** | the coordinate at which the axis line is to be drawn.<br>(i.e., the value on the other axis where it crosses) |
| **lty** | line type |
| **col** | the line and tick mark color |
| **las** | labels are parallel (=0) or perpendicular(=2) to axis |
| **tck** | length of tick mark as fraction of plotting region (negative number is outside graph, positive number is inside, 0 suppresses ticks, 1 creates gridlines) default is -0.01 |
| (...) | other graphical parameters |

# Graphical Parameters

```
# An Axis Example
# specify the data
x <- c(1:10); y <- x; z <- 10/x
# create extra margin room on the right for an axis
par(mar=c(5, 4, 4, 8) + 0.1)
# plot x vs. y
plot(x, y,type="b", pch=21, col="red", yaxt="n", lty=3, xlab="", ylab="")
# add x vs. 1/x
lines(x, z, type="b", pch=22, col="blue", lty=2)
# draw an axis on the left
axis(2, at=x,labels=x, col.axis="red", las=2)
# draw an axis on the right, with smaller text and ticks
axis(4, at=z,labels=round(z,digits=2),col.axis="blue", las=2, cex.axis=0.7, tck=-.01)
# add a title for the right axis
mtext("y=1/x", side=4, line=3, cex.lab=1,las=2, col="blue")
# add a main title and bottom and left axis labels
title("An Example of Creative Axes", xlab="X values", ylab="Y=X")
```

# Graphical Parameters

Reference Lines

Add reference lines to a graph using the abline( ) function.

abline(h=*yvalues*, v=*xvalues*)

Other ***graphical parameters*** (such as line type, color, and width) can also be specified in the abline( ) function.

```
# add solid horizontal lines at y=1,5,7
abline(h=c(1,5,7))
# add dashed blue verical lines at x = 1,3,5,7,9
abline(v=seq(1,10,2),lty=2,col="blue")
```

Note: You can also use the grid( ) function to add reference lines.

# Graphical Parameters

- **Legend**
- Add a legend with the **legend()** function.
- legend(location, title, legend, ...)
- Common options are described below.

| option | description |
|---|---|
| location | There are several ways to indicate the location of the legend. You can give an **x,y coordinate** for the upper left hand corner of the legend. You can use **locator(1)**, in which case you use the mouse to indicate the location of the legend. You can also use the **keywords** "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "bottomright", or "center". If you use a keyword, you may want to use **inset=** to specify an amount to move the legend into the graph (as fraction of plot region). |
| title | A character string for the legend title (optional) |
| legend | A character vector with the labels |
| ... | Other options. If the legend labels colored lines, specify **col=** and a vector of colors. If the legend labels point symbols, specify **pch=** and a vector of point symbols. If the legend labels line width or line style, use **lwd=** or **lty=** and a vector of widths or styles. To create colored boxes for the legend (common in bar, box, or pie charts), use **fill=** and a vector of colors. |

# Graphical Parameters

```
# Legend Example
attach(mtcars)
boxplot(mpg~cyl, main="Milage by Car Weight",
   yaxt="n", xlab="Milage", horizontal=TRUE,
   col=terrain.colors(3))
legend("topright", inset=.05, title="Number of
Cylinders",
   c("4","6","8"), fill=terrain.colors(3), horiz=TRUE)
```

# Graphical Parameters

- **Combining Plots**
- **R** makes it easy to combine multiple plots into one overall graph, using either the **par( )** or **layout( )** function.
- With the **par( )** function, you can include the option **mfrow=c(*nrows, ncols*)** to create a matrix of *nrows x ncols* plots that are filled in by row. **mfcol=c(*nrows, ncols*)** fills in the matrix by columns.
- # 4 figures arranged in 2 rows and 2 columns

```
attach(mtcars)
par(mfrow=c(2,2))
plot(wt,mpg, main="Scatterplot of wt vs. mpg")
plot(wt,disp, main="Scatterplot of wt vs disp")
 hist(wt, main="Histogram of wt")
boxplot(wt, main="Boxplot of wt")
```

# Graphical Parameters

- Three figures arranged in 3 rows and 1 column

```
attach(mtcars)
par(mfrow=c(3,1))
hist(wt)
hist(mpg)
hist(disp)
```

# Graphical Parameters

- The **layout( )** function has the form **layout(*mat*)** where *mat* is a matrix object specifying the location of the N figures to plot.
- # One figure in row 1 and two figures in row 2

```
attach(mtcars)
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))
hist(wt)
hist(mpg)
hist(disp)
```

# Graphical Parameters

- Optionally, you can include widths= and heights= options in the **layout( )** function to control the size of each figure more precisely. These options have the form
  **widths=** a vector of values for the widths of columns
  **heights=** a vector of values for the heights of rows.
- Relative widths are specified with numeric values. Absolute widths (in centimetres) are specified with the **lcm()** function.
- # One figure in row 1 and two figures in row 2
  # row 1 is 1/3 the height of row 2
  # column 2 is 1/4 the width of the column 1

```
attach(mtcars)
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE),
  widths=c(3,1), heights=c(1,2))
hist(wt)
hist(mpg)
hist(disp)
```

# Thank You