Introduction to Computer Programming with R (FOR 6934)

Qing Zhao (qing.zhao@ufl.edu)

Daijiang Li (dli1@ufl.edu)

Denis Valle (drvalle@ufl.edu)

Class Eleven

Graphic Visualization: parameters and saving

Some general rules

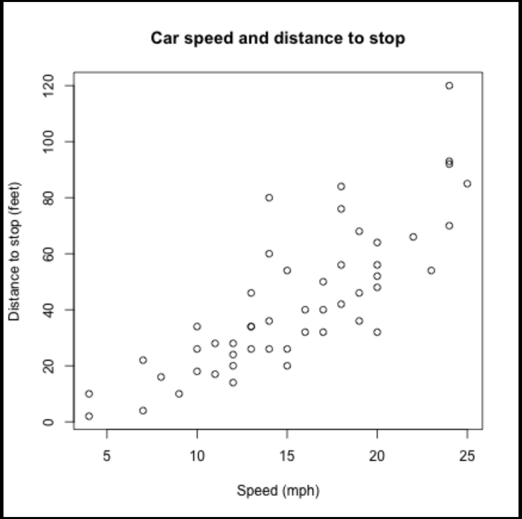
1. With Clear Purpose
2. Show the Data
3. Avoid Chart-junk
4. Utilize Data-ink
5. Utilize Color Carefully
6. Use Labels instead of Legend
7. Ease Comparisons
8. Separate Layers
9. Sort on meaningful variables
10. tinyurl.com/graphs2017

R is an amazing tool for creating graphs

Plot examples

Scatter plots

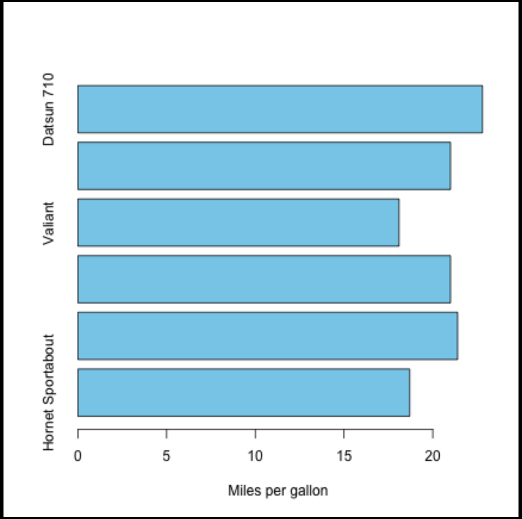
```r
data("cars")
plot(x = cars$speed,
     y = cars$dist,
     type = "p",
     main = "Car speed and distance to stop",
     xlab = "Speed (mph)",
     ylab = "Distance to stop (feet)")
```



Plot examples

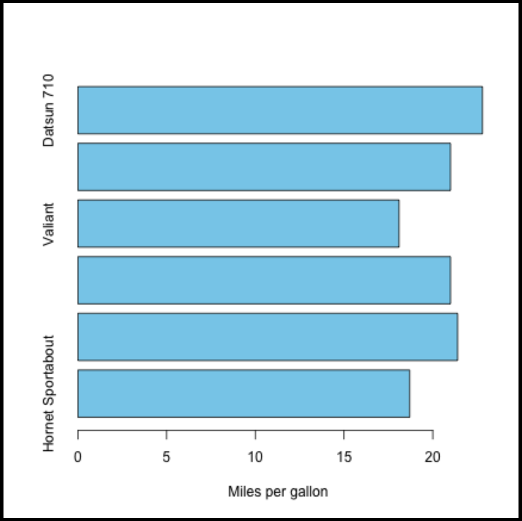
Barplot

```
data("mtcars")
d = mtcars[sample(6),]
barplot(d$mpg,
        xlab = "Miles per gallon",
        col = "skyblue",
        horiz = T,
        names.arg = row.names(d))
```



Plot examples

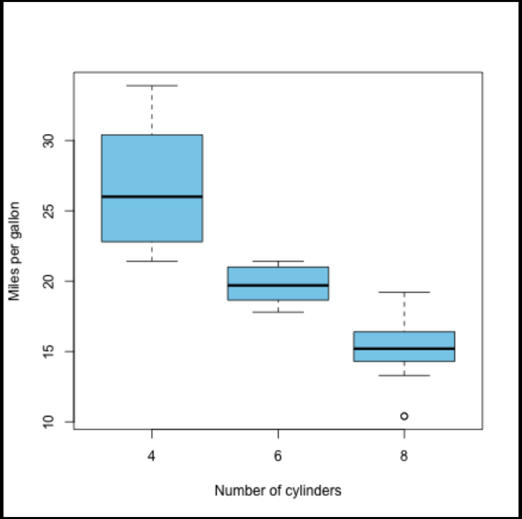
How to improve this plot??



Plot examples

Boxplot

```
boxplot(mtcars$mpg ~ mtcars$cyl,
        xlab = "Number of cylinders",
        ylab = "Miles per gallon",
        col = "skyblue")
```



Plot examples

Pause the video and type `demo(graphics)` in R to convince yourself

Plot examples

Pause the video and type `demo(graphics)` in R to convince yourself

More examples: here, here, here, and here,

Main packages for plotting

Build-in packages: `graphics` and `lattice`

Other popular packages: `ggplot2`, `plotly` (interactive plots)

Common functions from the `graphics` package

- `plot`
- `points`
- `lines`
- `hist`
- `density`
- `boxplot`
- `barplot`
- `dotchart`
- `stripchart`
- `pairs`
- `xplot`
- `image`
- `contour`
- `legend`
- `arrows`
- `abline`

A cheatsheet may be useful.

This concludes Class 11, Section 1

Please continue on to the next video

Customizing graphics

Customizing graphics

1. Changing arguments within a charting function

Customizing graphics

1. Changing arguments within a charting function
2. Changing global graphic parameters via `par()`

Common within function arguments to customize charts[1]

Argument	Description
add	add to the existing plot? `plot(..., add = TRUE)`
axes	plot axes? `plot(..., axes = FALSE)`
log	points plot on a logarithmic scale? `plot(..., log ="xy")`
type	type of graph being plotted. `plot(..., type ="p")`
xlab, ylab	labels of x- and y-axes. `plot(..., xlab ="x lab")`
main	main title for the plot. `plot(..., main ="main title")`
sub	subtitle for the plot. `plot(..., sub ="sub title")`

[1]: in doubt, read the documentation by typing `?plot_function`

Common within function arguments to customize charts (cont.)

Argument	Description
col	symbol color. e.g. `plot(..., col ="red")`, `col.axis`, `col.lab`, `col.main`, etc.
pch	symbol styles. `plot(..., pch = 16)`
cex	symbol size. `plot(..., cex = 2)`, `cex.main`, `cex.lab`, etc.
lty	line type: 0-6. blank, solid (default), dashed, etc. `plot(..., lty = 2)`
lwd	line width. `plot(..., lwd = 2)`

Common within function arguments to customize charts (cont.)

Argument	Description
col	symbol color. e.g. `plot(..., col ="red")`, `col.axis`, `col.lab`, `col.main`, etc.
pch	symbol styles. `plot(..., pch = 16)`
cex	symbol size. `plot(..., cex = 2)`, `cex.main`, `cex.lab`, etc.
lty	line type: 0-6. blank, solid (default), dashed, etc. `plot(..., lty = 2)`
lwd	line width. `plot(..., lwd = 2)`

For most parameters (but not all), you can set them as arguments to graphics functions.

You can also set them (and all) via the `par()` function.

Try to change some of the arguments and run the code by yourself to see what happens:

```r
set.seed(123) # to be reproducible
plot(x = 1:100, y = rnorm(100),
     pch = 16, cex = 1.2, col = "blue",
     type = "b", # "p", "l", "o", "s", "h"
     lty = 2, lwd = 1.2,
     log = "", # "x", "y", "xy"
     xlab = "x", ylab = "y",
     main = "A scatter plot",
     sub = "mini example")
text(x = 5, y = -2.2, labels = "A point",
     pos = 4)
```

Global graphical parameters via `par()`

Global graphical parameters via `par()`

By setting global parameters, the new settings will be applied to every new plot

Global graphical parameters via `par()`

By setting global parameters, the new settings will be applied to every new plot

To check all values of these parameters, type `par()`

Global graphical parameters via `par()`

By setting global parameters, the new settings will be applied to every new plot

To check all values of these parameters, type `par()`

```
names(par()) # truncated output
```

```
##  [1] "xlog"      "ylog"      "adj"       "ann"       "ask"
##  [6] "bg"        "bty"       "cex"       "cex.axis"  "cex.lab"
## [11] "cex.main"  "cex.sub"   "cin"       "col"       "col.axis"
## [16] "col.lab"   "col.main"  "col.sub"   "cra"       "crt"
## [21] "csi"       "cxy"       "din"       "err"       "family"
## [26] "fg"        "fig"       "fin"       "font"      "font.axis"
## [31] "font.lab"  "font.main" "font.sub"  "lab"       "las"
## [36] "lend"      "lheight"   "ljoin"     "lmitre"    "lty"
## [41] "lwd"       "mai"       "mar"       "mex"       "mfcol"
## [46] "mfg"       "mfrow"     "mgp"       "mkh"       "new"
## [51] "oma"       "omd"       "omi"       "page"      "pch"
## [56] "pin"       "plt"       "ps"        "pty"       "smo"
## [61] "srt"       "tck"       "tcl"       "usr"       "xaxp"
```

Global graphical parameters via `par()`

To check one parameter, use the name of the parameter as argument

```r
par("bg")
```

```
## [1] "white"
```

Global graphical parameters via `par()`

To check one parameter, use the name of the parameter as argument

```r
par("bg")
```

```
## [1] "white"
```

To change parameter(s), specify new value(s)

```r
par(bg = "black", fg = "white")
```

Global graphical parameters via `par()`

To check one parameter, use the name of the parameter as argument

```
par("bg")
```

```
## [1] "white"
```

To change parameter(s), specify new value(s)

```
par(bg = "black", fg = "white")
```

Almost all parameters can be changed (except read-only ones: `cin, cra, csi, cxy, din, page`)

`?par` is your friend*

*: other friends: `?regex`, `?plotmath`. Read these pages multiple times!

`?par` is your friend*

I will only give examples about how to change some commonly used parameters

*: other friends: `?regex`, `?plotmath`. Read these pages multiple times!

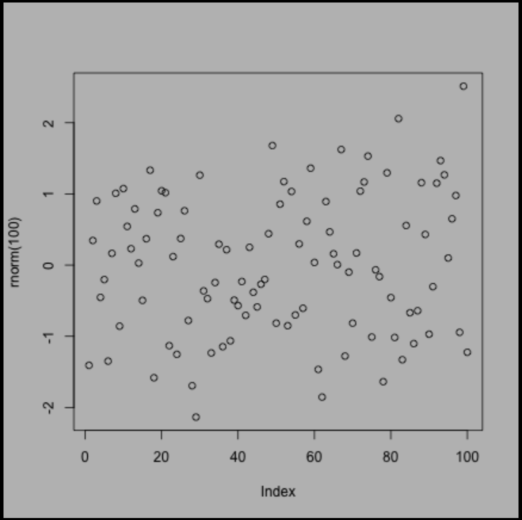
Save, change, and restore the default setting

```r
# save original values of changeable parameters
original_par <- par(no.readonly = TRUE)
# change
par(lty = 2, pch = 17)
pch(col = "blue") # can be separate calls
# plot
hist(mtcars$mpg)
# then restore
par(original_par)
```

This is a good habit.

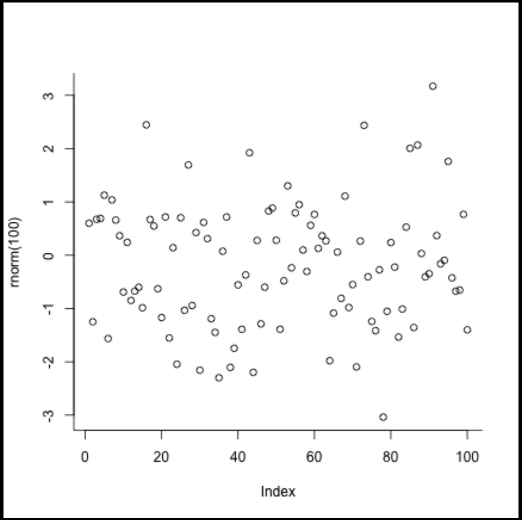
Setting plot background colors: `bg`

```
par(bg = "gray")
plot(rnorm(100))
```

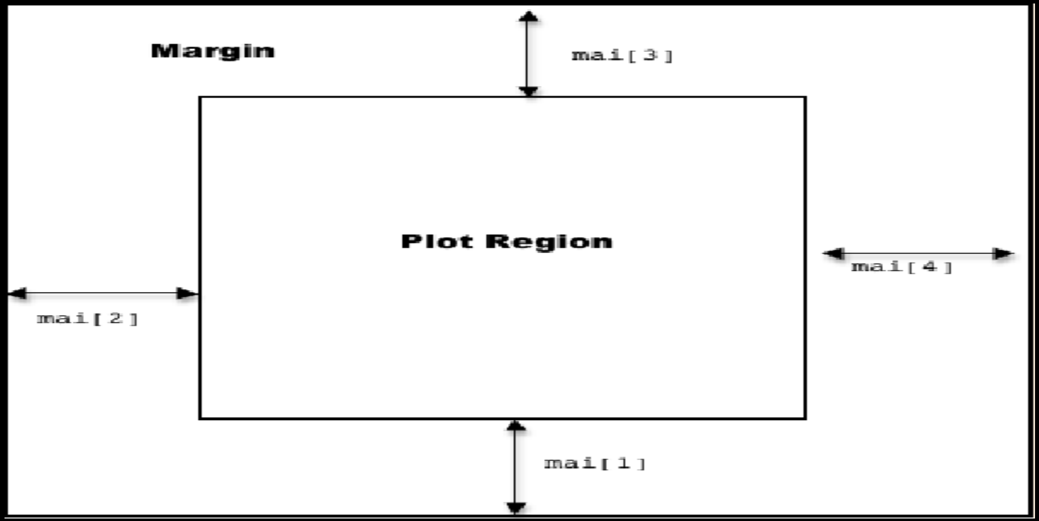


Choosing box styles: `bty`

```r
par(bty = "l")
# other values: 1, o, c, u, 7,  n (no),
# and right square bracket
plot(rnorm(100))
```

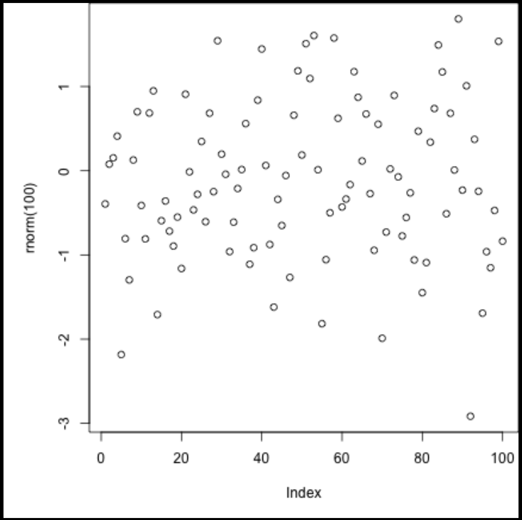


Setting plot margins: `mai` or `mar`



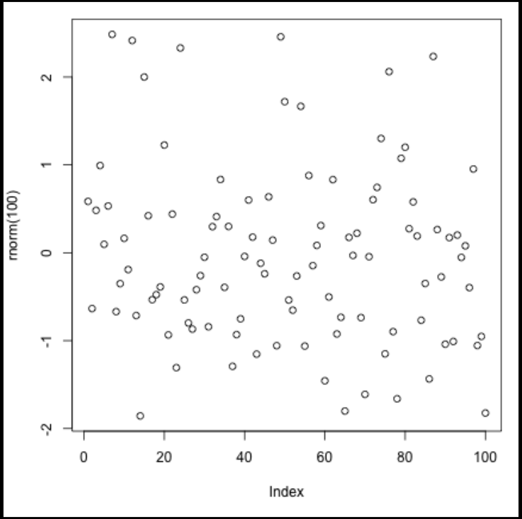
Setting plot margins: `mai`

```
par(mai = c(1, 1, 0, 0))
# c(botton, left, top, right)
# MArgin size in Inches
plot(rnorm(100))
```



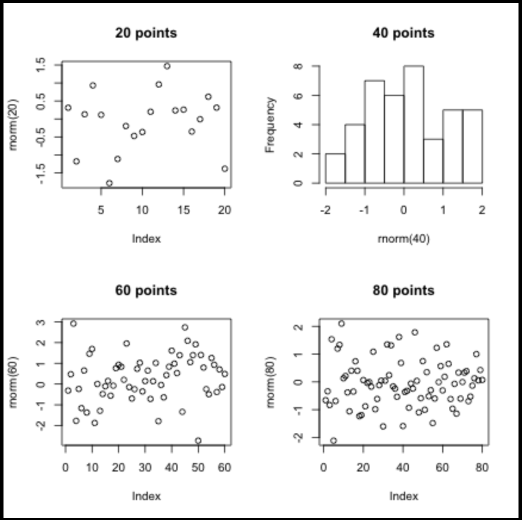
Setting plot margins: `mar`

```
par(mar = c(5, 4, 1, 1))
# c(botton, left, top, right)
# MARgin size in number of lines
plot(rnorm(100))
```



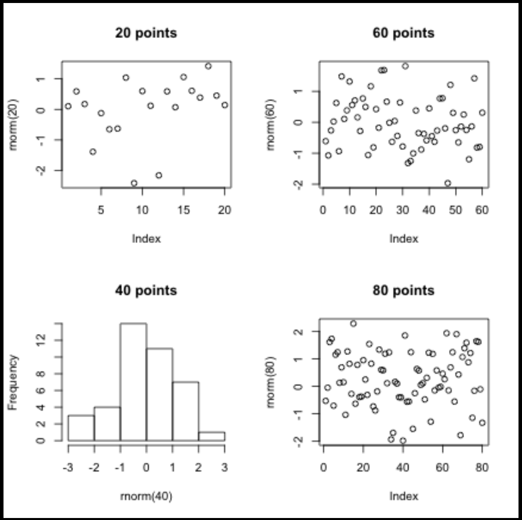
Creating multiple plots in one figure: `mfrow` or `mfcol`

```r
par(mfrow = c(2, 2))
plot(rnorm(20), main = "20 points")
hist(rnorm(40), main = "40 points")
plot(rnorm(60), main = "60 points")
plot(rnorm(80), main = "80 points")
```



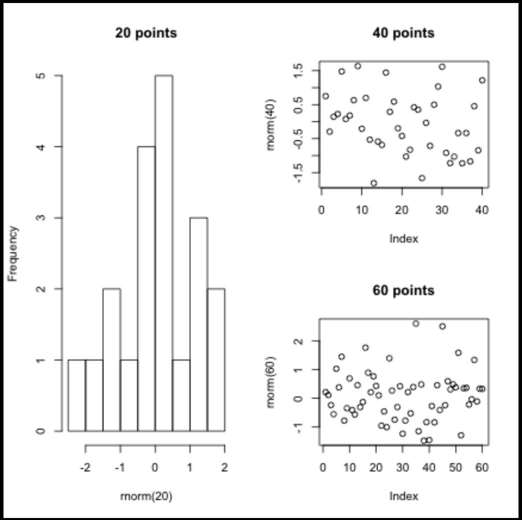
Creating multiple plots in one figure: `mfrow` or `mfcol`

```r
par(mfcol = c(2, 2))
plot(rnorm(20), main = "20 points")
hist(rnorm(40), main = "40 points")
plot(rnorm(60), main = "60 points")
plot(rnorm(80), main = "80 points")
```



Creating multiple plots in one figure: `layout()`

```r
layout(matrix(c(1, 1, 2, 3),
              nrow = 2, ncol = 2))
hist(rnorm(20), main = "20 points")
plot(rnorm(40), main = "40 points")
plot(rnorm(60), main = "60 points")
?layout # for more details
```



This concludes Class 11, Section 2

Please continue on to the next video

Save graphics

Saving Graphics: general points

Grahpics in R are plotted on a *graphics device*

- `windows` on Windows OS
- `X11` on Unix systems
- `quartz` on macOS

Saving Graphics: general points

Grahpics in R are plotted on a *graphics device*

- `windows` on Windows OS
- `X11` on Unix systems
- `quartz` on macOS

You can save graphics in common formats

- `png`, `jpeg`, `bmp`, `tiff`, `pdf`, `svg`, `postscript`, `pictex`, `xfig`, etc.
- recommend to save as vector graphics such as PDF and SVG

Saving Graphics: general points

Grahpics in R are plotted on a *graphics device*

- `windows` on Windows OS
- `X11` on Unix systems
- `quartz` on macOS

You can save graphics in common formats

- `png`, `jpeg`, `bmp`, `tiff`, `pdf`, `svg`, `postscript`, `pictex`, `xfig`, etc.
- recommend to save as vector graphics such as PDF and SVG

You can specify output width, height, and point size when saving graphics

Saving Graphics: general workflow

```
FORMAT_NAME_WANTED(filename = "fig_path_name.EXTENSION",
                      width = 4, height = 4, units = "in")
# 1. code to set parameters (optional)
# 2. code to load data (optional)
# 3. code to plot figure(s)
# 4. code to refine the figure (optional)
dev.off()
```

Saving Graphics: examples (copy and run by yourself)

```r
png(filename = "fig_path_name.png", type = "cairo",
       width = 4, height = 4, units = "in", res = 100)
# 1. code to set parameters
par(bg = "yellow")
# 2. code to load data
data("mtcars")
# 3. code to plot figure
plot(mtcars$hp, mtcars$mpg, pch = 16)
# 4. code to refine the figure
abline(lm(mpg ~ hp, data = mtcars))
dev.off()
```

Saving Graphics: examples (copy and run by yourself)

```r
png(filename = "fig_path_name.png", type = "cairo",
       width = 4, height = 4, units = "in", res = 100)
# 1. code to set parameters
par(bg = "yellow")
# 2. code to load data
data("mtcars")
# 3. code to plot figure
plot(mtcars$hp, mtcars$mpg, pch = 16)
# 4. code to refine the figure
abline(lm(mpg ~ hp, data = mtcars))
dev.off()
```

With increasing resolution, image format and layout may change and you need to adjust for that

`type = "cairo"` can be helpful in conserving point size

Saving Graphics: examples (copy and run by yourself)

```r
pdf(file = "fig_path_name.pdf", width = 4, height = 4,
       pointsize = 12, onefile = TRUE, colormodel = "cmyk")
par(mfrow = c(1, 2)) # 1. code to set parameters
data(trees) # 2. code to load data
boxplot(trees$Girth, main = "Girth boxplot")
hist(trees$Girth, main = "Girth historgram")
par(mfrow = c(1, 1)) # reset par
plot(trees$Height, trees$Volume, pch = 16, col = "blue")
dev.off() # a pdf with TWO pages
```

`cmyk`: Cyan Magenta Yellow Key. Default is `srgb` (sRGB).

Most publications require authors to use `cmyk` in graphs.

Saving Graphics: examples (copy and run by yourself)

```
svg(file = "fig_path_name.svg", width = 6, height = 3, pointsize = 12)
par(mfrow = c(1, 2)) # 1. code to set parameters
data(trees) # 2. code to load data
boxplot(trees$Girth, main = "Girth boxplot")
hist(trees$Girth, main = "Girth historgram")
dev.off()
```

Install `Cario` package: `install.packages("Cario")`

Windows users need to use `Cario::CarioSVG()` instead of `svg()`

Summary

So many plot functions and graphics parameters

Functions can be used sequentially to build graphics

Read the documentations & Google

Practice

Thank you and see you next class