

# Graphics in R

## STAT 133

Gaston Sanchez

`github.com/ucb-stat133/stat133-fall-2016`

# Base Graphics

# Graphics in R

## Traditional Graphics

- ▶ R "graphics" follows a static, "painting on canvas" model.
- ▶ Graphics elements are drawn, and remain visible until painted over.
- ▶ For dynamic and/or interactive graphics, R is limited.

# Traditional Graphics

## Traditional graphics model

In the traditional model, we create a plot by first calling a high-level function that creates a complete plot, and then we call low-level functions to add more output if necessary

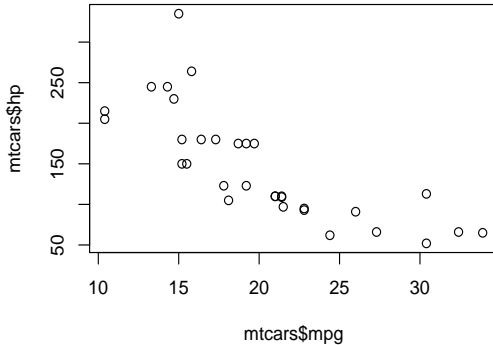
# Dataset mtcars

```
head(mtcars)
```

##	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
## Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
## Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
## Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
## Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
## Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
## Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

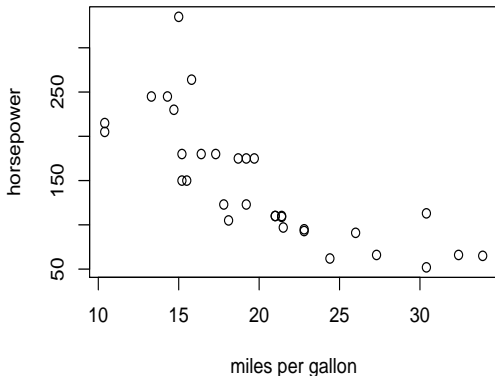
# Scatter plot

```
# simple scatter-plot  
plot(mtcars$mpg, mtcars$hp)
```



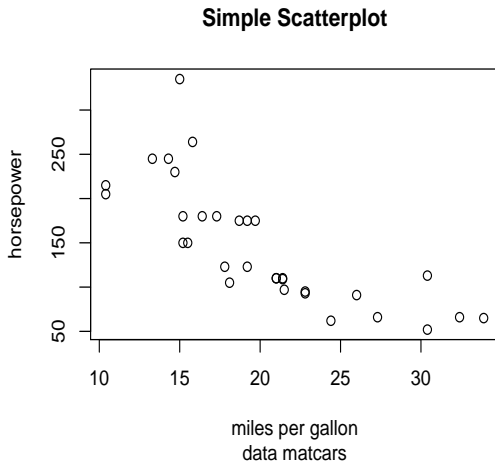
# Axis Labels

```
# xlab and ylab  
plot(mtcars$mpg, mtcars$hp, xlab = "miles per gallon",  
      ylab = "horsepower")
```



# Title and subtitle

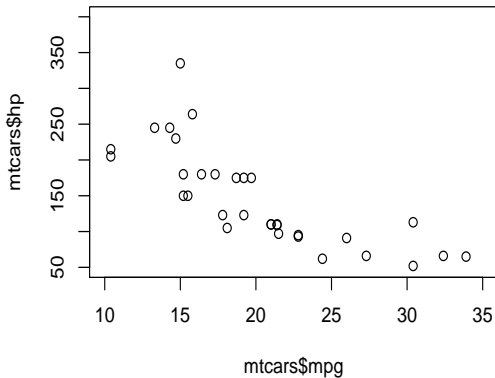
```
# title and subtitle  
plot(mtcars$mpg, mtcars$hp, xlab = "miles per gallon",  
     ylab = "horsepower", main = "Simple Scatterplot",  
     sub = 'data mtcars')
```





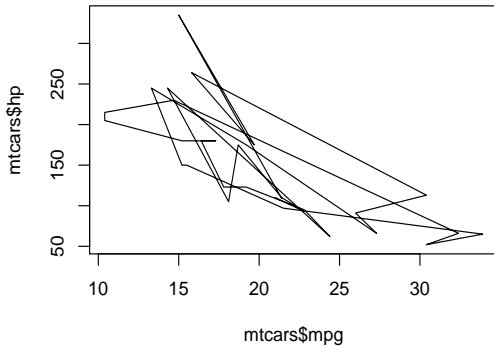
# x and y coordinate ranges

```
# 'xlim' and 'ylim'  
plot(mtcars$mpg, mtcars$hp, xlim = c(10, 35), ylim = c(50, 400))
```



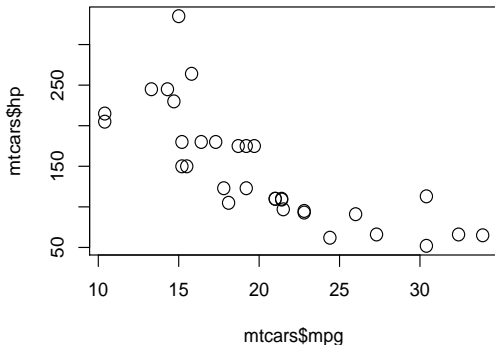
# Type

```
# using 'type' (e.g. lines)  
plot(mtcars$mpg, mtcars$hp, type = "l")
```




























# Points

```
# character expansion 'cex'  
# and 'point character'  
plot(mtcars$mpg, mtcars$hp, cex = 1.5, pch = 1)
```

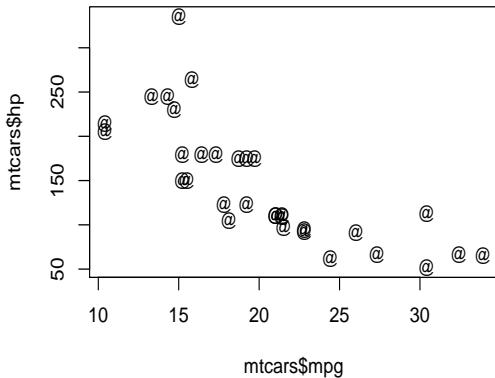


# Point symbols (pch) available in R

 1	 2	 3	 4	 5
 6	 7	 8	 9	 10
 11	 12	 13	 14	 15
 16	 17	 18	 19	 20
 21	 23	 24	 25	 26

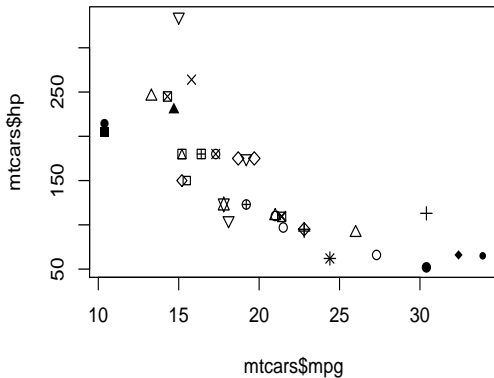
# Point Character

```
# 'pch' can be any character  
plot(mtcars$mpg, mtcars$hp, pch = "@")
```



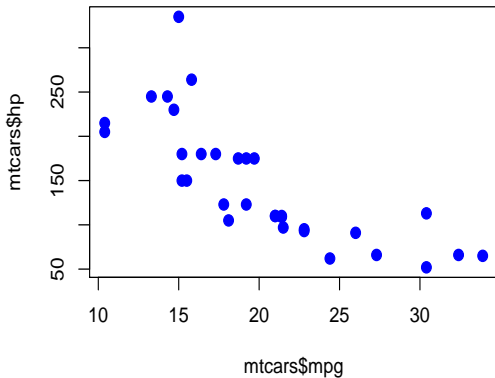
# Point Character

```
# 'pch' symbols will be recycled  
plot(mtcars$mpg, mtcars$hp, pch = 1:25)
```



# Point Colors

```
# color argument 'col'  
plot(mtcars$mpg, mtcars$hp, pch = 19, col = "blue", cex = 1.2)
```



# Coloring Point Symbols

- ▶ the `col` argument can be used to color symbols
- ▶ symbols 21 through 25 can additionally have their interiors filled by using the `bg` (background) argument



# Coloring Point symbols



1



2



3



4



5



6



7



8



9



10



11



12



13



14



15



16



17



18



19



20



21



22



23



24

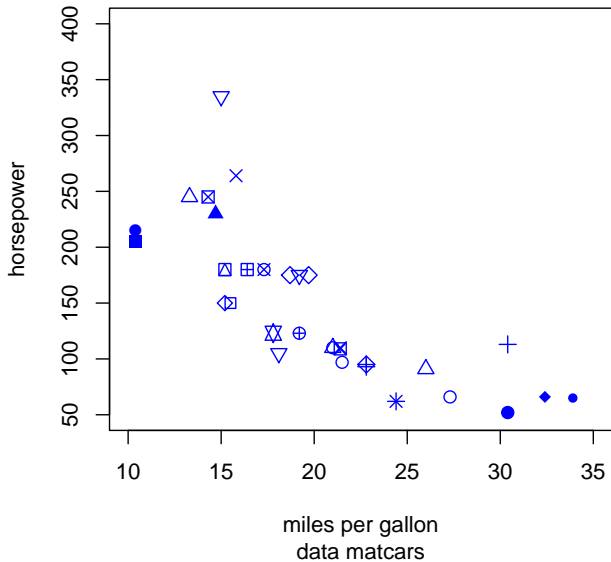


25

# So far ...

```
# using plot()  
plot(mtcars$mpg, mtcars$hp,  
      xlim = c(10, 35), ylim = c(50, 400),  
      xlab = "miles per gallon",  
      ylab = "horsepower",  
      main = "Simple Scatterplot",  
      sub = 'data mtcars',  
      pch = 1:25, cex = 1.2, col = "blue")
```

## Simple Scatterplot



# Low-Level Functions

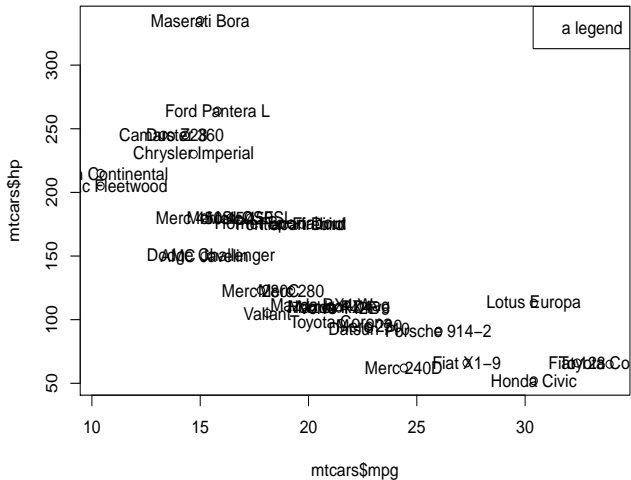
# High and Low level functions

- ▶ Usually we call a high-level function
- ▶ Most times we change the default arguments
- ▶ Then we call low-level functions

# Scatter plot

```
# simple scatter-plot  
plot(mtcars$mpg, mtcars$hp)  
  
# adding text  
text(mtcars$mpg, mtcars$hp, labels = rownames(mtcars))  
  
# dummy legend  
legend("topright", legend = "a legend")  
  
# graphic title  
title("Miles Per Galon -vs- Horsepower")
```

## Miles Per Gallon –vs– Horsepower

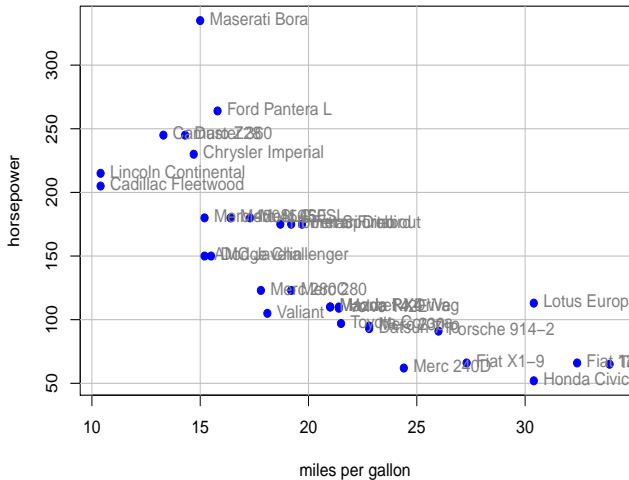


# Scatter plot

```
# simple scatter-plot
plot(mtcars$mpg, mtcars$hp, type = "n",
      xlab = "miles per gallon", ylab = "horsepower")
# grid lines
abline(v = seq(from = 10, to = 30, by = 5), col = 'gray')
abline(h = seq(from = 50, to = 300, by = 50), col = ' gray')
# plot points
points(mtcars$mpg, mtcars$hp, pch = 19, col = "blue")
# plot text
text(mtcars$mpg, mtcars$hp, labels = rownames(mtcars),
      pos = 4, col = "gray50")
# graphic title
title("Miles Per Galon -vs- Horsepower")
```



# Miles Per Gallon –vs– Horsepower



# Test your knowledge

## The function `plot()`

- A) accepts any type of vector
- B) is a generic function
- C) works only for 1-D and 2-D objects
- D) is designed to display scatterplots and boxplots

## Low-level functions

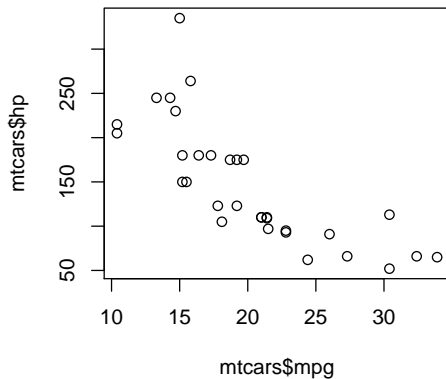
Function	Description
<code>points()</code>	points
<code>lines()</code>	connected line segments
<code>abline()</code>	straight lines across a plot
<code>segments()</code>	disconnected line segments
<code>arrows()</code>	arrows
<code>rect()</code>	rectangles
<code>polygon()</code>	polygons
<code>text()</code>	text
<code>symbols()</code>	various symbols
<code>legend()</code>	legends

# Drawing Points with `points()`

```
points(x, y, pch = int, col = str)
```

- ▶ `pch` integer or string indicating type of point character
- ▶ `col` color of points

```
# drawing points  
plot(mtcars$mpg, mtcars$hp, type = "n")  
points(mtcars$mpg, mtcars$hp)
```

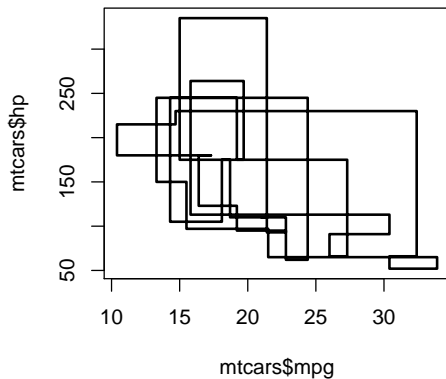


# Connected Line Segments

```
lines(x, y, lty = str, lwd = num, col = str)
```

- ▶ `lty` specifies the line texture. It should be one of "blank" (0), "solid" (1), "dashed" (2), "dotted" (3), "dotdash" (4), "longdash" (5) or "twodash" (6).
- ▶ `lwd` and `col` specify the line width and colour

```
# connected lines  
plot(mtcars$mpg, mtcars$hp, type = "n")  
lines(mtcars$mpg, mtcars$hp, type = "s", lwd = 2)
```



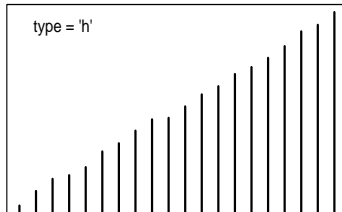
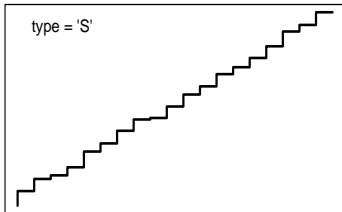
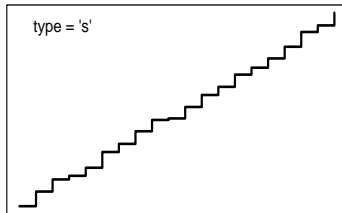
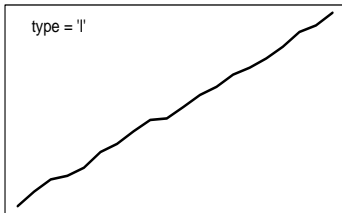
# Line Graph Options

The type argument can be used to produced other types of lines

- ▶ `type = "l"` line graph
- ▶ `type = "s"` step function (horizontal first)
- ▶ `type = "S"` step function (vertical first)
- ▶ `type = "h"` high density (needle) plot
- ▶ `type = "p"` draw points
- ▶ `type = "b"` draw points and lines
- ▶ `type = "o"` over-plotting points and lines



# Line Graph Options



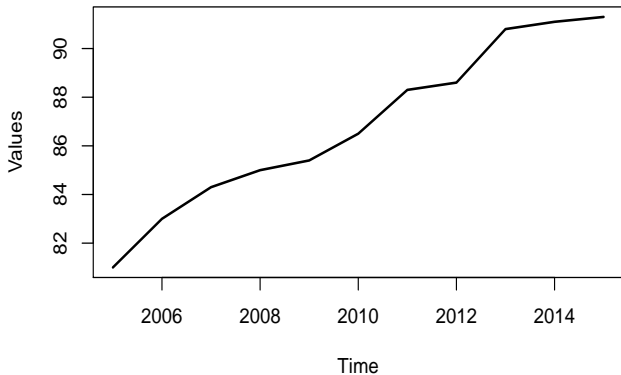
# Connected Line Segments

```
x <- 2005:2015
y <- c(81, 83, 84.3, 85, 85.4, 86.5, 88.3,
      88.6, 90.8, 91.1, 91.3)

plot(x, y, type = 'n', xlab = "Time", ylab = "Values")
lines(x, y, lwd = 2)
title(main = "Line Graph Example")
```

# Connected Line Segments

**Line Graph Example**



# Drawing Straight Lines

```
abline(a = intercept, b = slope)
```

```
abline(h = numbers)
```

```
abline(v = numbers)
```

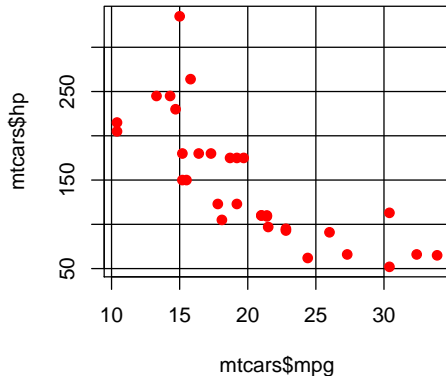
- ▶ The  $a / b$  form specifies a line in intercept / slope form
- ▶  $h$  specifies horizontal lines at given y-values
- ▶  $v$  specifies vertical lines at given x-values

```
# drawing straight lines
```

```
plot(mtcars$mpg, mtcars$hp, type = "n")
```

```
abline(v = seq(10, 30, by = 5), h = seq(50, 300, by = 50))
```

```
points(mtcars$mpg, mtcars$hp, pch = 19, col = "red")
```



# Drawing Disconnected Lines

Disconnected lines can be drawn with the function:

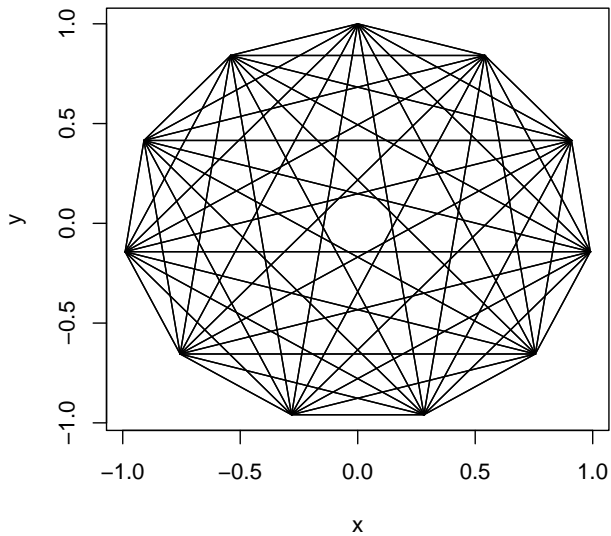
```
segments(x0, y0, x1, y1)
```

- ▶ The `x0`, `y0`, `x1`, `y1` arguments give the start and end coordinates of the segments.
- ▶ Line texture, colour and width arguments can also be given.

# Drawing Line Segments

```
n <- 11
theta <- seq(0, 2 * pi, length = n + 1)[1:n]
x <- sin(theta)
y <- cos(theta)
v1 <- rep(1:n, n)
v2 <- rep(1:n, rep(n, n))

plot(x, y, type = 'n')
segments(x[v1], y[v1], x[v2], y[v2])
```





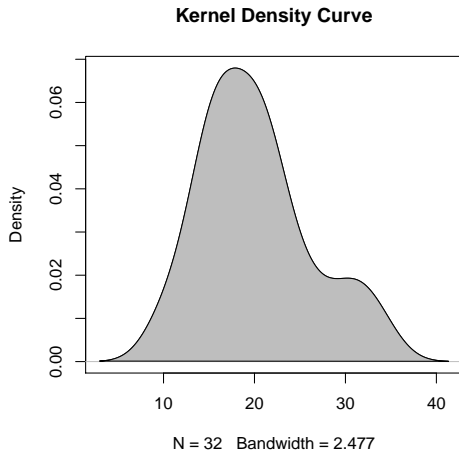
# Drawing Polygons

Polygons can be drawn with the function:

```
polygon(x, y, col = str, border = str)
```

- ▶ `x`, `y` give the coordinates of the polygon vertexes. NA values separate polygons.
- ▶ `col` specifies the color of the interior.
- ▶ `border` specifies the color of the border.
- ▶ line texture and width specifications can also be given

# Drawing Polygons



# Adding Text

We can add text using the function:

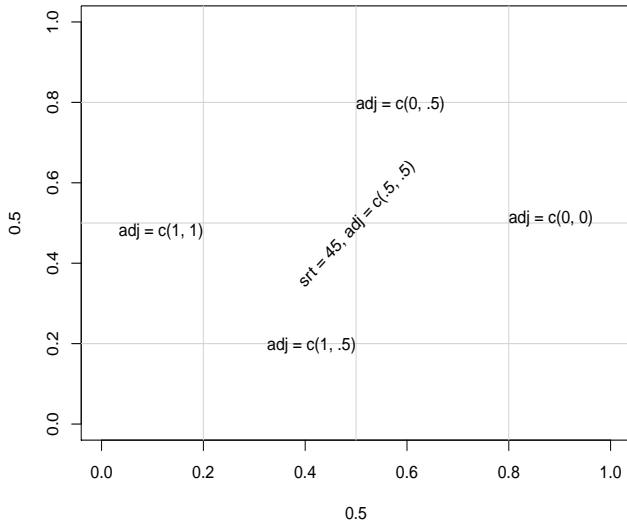
```
text(x, y, labels, ...)
```

- ▶ `x`, `y` give the coordinates of the text.
- ▶ `labels` gives the actual text strings.
- ▶ `font` optional font for the text.
- ▶ `col` optional color for the text.
- ▶ `srt` rotation of the text.
- ▶ `adj` justification of the text.

# Drawing Text

```
plot(0.5, 0.5, xlim = c(0, 1), ylim = c(0, 1), type = 'n')
abline(h = c(.2, .5, .8),
       v = c(.5, .2, .8), col = "lightgrey")
text(0.5, 0.5, "srt = 45, adj = c(.5, .5)",
     srt = 45, adj = c(.5, .5))
text(0.5, 0.8, "adj = c(0, .5)", adj = c(0, .5))
text(0.5, 0.2, "adj = c(1, .5)", adj = c(1, .5))
text(0.2, 0.5, "adj = c(1, 1)", adj = c(1, 1))
text(0.8, 0.5, "adj = c(0, 0)", adj = c(0, 0))
```

# Drawing Text



# Adding a Legend

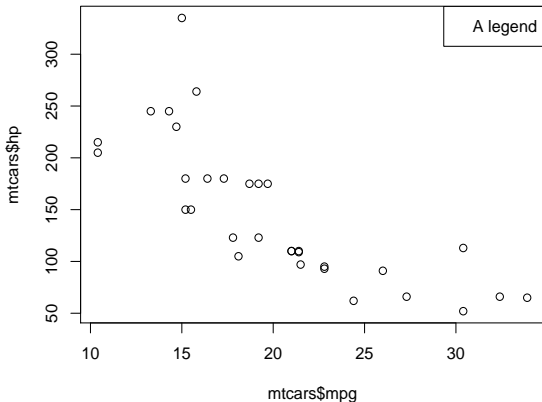
Legends can be added with:

```
legend(xloc, yloc, legend = text  
      lty = linetypes, lwd = linewidths,  
      pch = glyphname, col = colours,  
      xjust = justification, yjust = justification)
```

- ▶ `xloc` and `yloc` give the coordinates where the legend is to be placed
- ▶ `xjust` and `yjust` give the justification of the legend box with respect to the location.

# Adding Legends

```
# coords of exact line  
plot(mtcars$mpg, mtcars$hp)  
legend("topright", legend = "A legend")
```



# Plots from scratch



# Customizing Annotations

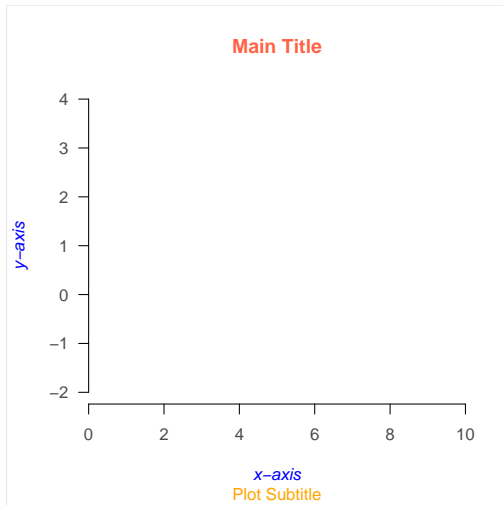
It is also possible to create a plot from scratch. Although this procedure is less documented, it is extremely flexible:

1. call `plot.new()` to start a new plot frame
2. call `plot.window()` to define coordinates
3. then call low-level functions:
4. typical options involve `axis()`
5. then `title()` (title, subtitle)
6. after that call other function: e.g. `points()`, `lines()`, etc

# Plot from Scratch

```
plot.new()
plot.window(xlim = c(0, 10), ylim = c(-2, 4), xaxs = "i")
axis(1, col.axis = "grey30")
axis(2, col.axis = "grey30", las = 1)
title(main = "Main Title",
      col.main = "tomato",
      sub = "Plot Subtitle",
      col.sub = "orange",
      xlab = "x-axis", ylab = "y-axis",
      col.lab = "blue", font.lab = 3)
box("figure", col = "grey90")
```

# Plot from Scratch

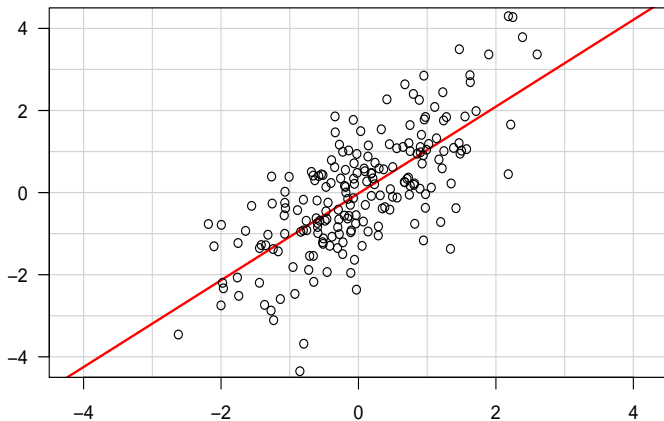


## Another plot from scratch

```
set.seed(5)
x <- rnorm(200)
y <- x + rnorm(200)

plot.new()
plot.window(xlim = c(-4.5, 4.5), xaxs = "i",
            ylim = c(-4.5, 4.5), yaxs = "i")
z <- lm(y ~ x)
abline(h = -4:4, v = -4:4, col = "lightgrey")
abline(a = coef(z)[1], b = coef(z)[2], lwd = 2, col = "red")
points(x, y)
axis(1)
axis(2, las = 1)
box()
title(main = "A Fitted Regression Line")
```

**A Fitted Regression Line**



# Creating a Plot from Scratch

- ▶ Start a new plot with `plot.new()`
- ▶ `plot.new()` opens a new (empty) plot frame
- ▶ `plot.new()` chooses a default plotting region

# Setting Up Coordinates

Then use `plot.window()` to set up the coordinate system for the plotting frame

```
# axis limits (0,1)x(0,1)  
plot.window(xlim = c(0, 1), ylim = c(0, 1))
```

By default `plot.window()` produces axis limits which are expanded by 6% over those actually specified.

# Setting Up Coordinates

The default limits expansion can be turned-off by specifying `xaxs = "i"` and/or `yaxs = "i"`

```
plot.window(xlim, ylim, xaxs = "i")
```



# Aspect Ratio Control

Another important argument is `asp`, which allows us to specify the **aspect ratio**

```
plot.window(xlim, ylim, xaxs = "i", asp = 1)
```

`asp = 1` means that unit steps in the x and y directions produce equal distances in the x and y directions on the plot. (Important to avoid distortion of circles that look like ellipses)

# Drawing Axes

The `axis()` function can be used to draw axes at any of the four sides of a plot.

- ▶ `side=1` below the graph
- ▶ `side=2` to the left of the graph
- ▶ `side=3` above the graph
- ▶ `side=4` to the right of the graph

# Customizing Axes

Axes can be customized via several arguments (see `?axis`)

- ▶ location of tick-marks
- ▶ labels of axis
- ▶ colors
- ▶ sizes
- ▶ text fonts
- ▶ text orientation

# Plot Annotation

The function `title()` allows us to include labels in the margins

- ▶ `main` main title above the graph
- ▶ `sub` subtitle below the graph
- ▶ `xlab` label for the x-axis
- ▶ `ylab` label for the y-axis

# Customizing Annotations

The annotations can be customized with additional arguments for the fonts, colors, and size (expansion)

- ▶ `font.main`, `col.main`, `cex.main`
- ▶ `font.sub`, `col.sub`, `cex.sub`
- ▶ `font.lab`, `col.lab`, `cex.lab`

# Drawing Arrows

Arrows can be drawn with the function:

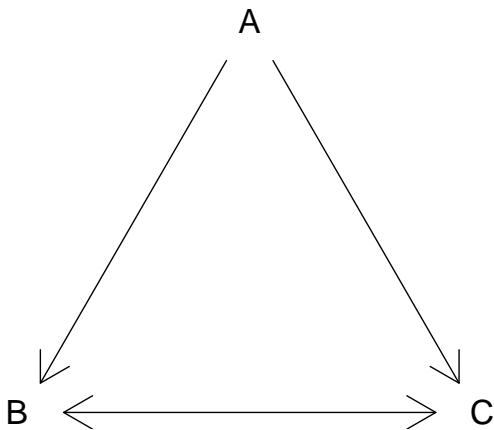
```
arrows(x0, y0, x1, y1, code = int,  
       length = num, angle = num)
```

- ▶ The `x0`, `y0`, `x1`, `y1` arguments give the start and end coordinates.
- ▶ `code=1` head at the start, `code=2` head at the end, `code=3` head at both ends
- ▶ `length` of the arrow head and `angle` to the shaft

# Drawing Arrows

```
plot.new()
plot.window(xlim = c(0, 1), ylim = c(0, 1))
arrows(.05, .075, .45, .9, code = 1)
arrows(.55, .9, .95, .075, code = 2)
arrows(.1, 0, .9, 0, code = 3)
text(.5, 1, "A", cex = 1.5)
text(0, 0, "B", cex = 1.5)
text(1, 0, "C", cex = 1.5)
```

# Drawing Arrows





# Drawing Rectangles

Rectangles can be drawn with the function:

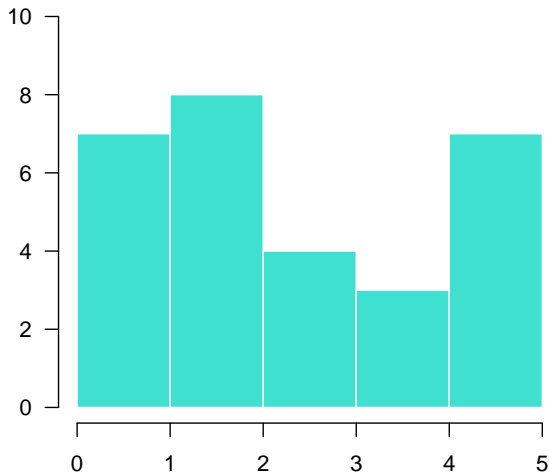
```
rect(x0, y0, x1, y1, col = str, border = str)
```

- ▶ `x0`, `y0`, `x1`, `y1` give the coordinates of diagonally opposite corners of the rectangles.
- ▶ `col` specifies the color of the interior.
- ▶ `border` specifies the color of the border.

# Drawing Rectangles

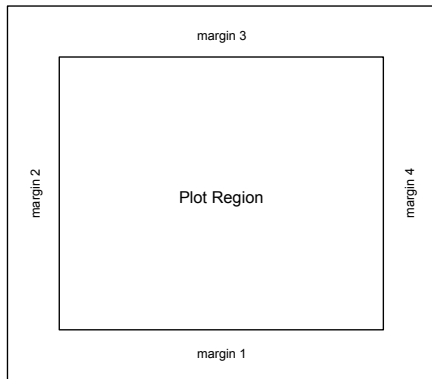
```
# barplot manually constructed
plot.new()
plot.window(xlim = c(0, 5), ylim = c(0, 10))
rect(0:4, 0, 1:5, c(7, 8, 4, 3),
     col = "turquoise",
     border = "white")
axis(1)
axis(2, las = 1)
```

# Drawing Rectangles



# Plot Regions

# Anatomy of Plot Frame and Region



# Adjusting the Margins

Margins can be adjusted with the `par()` function in various ways:

- ▶ In inches: `par(mai = c(2, 2, 1, 1))`
- ▶ In lines of text: `par(mar = c(4, 4, 2, 2))`
- ▶ Width and Height in inches: `par(pin = c(5, 4))`

# One more scatter plot

```
# simple scatter-plot
op <- par(mar = c(5, 4, 3, 1))
plot(mtcars$mpg, mtcars$hp, type = "n",
      xlab = "miles per gallon", ylab = "horsepower")
# grid lines
abline(v = seq(from = 10, to = 30, by = 5), col = 'gray')
abline(h = seq(from = 50, to = 300, by = 50), col = ' gray')
# points
points(mtcars$mpg, mtcars$hp, pch = 19, col = "blue")
# text (point labels)
text(mtcars$mpg, mtcars$hp, labels = rownames(mtcars),
      pos = 4, col = "gray50")
# title
title("Miles Per Galon -vs- Horsepower")
# reset graphical margins
par(op)
```

## Miles Per Galon –vs– Horsepower

