Bio Stats II : Lab 2

Acknowledgements: Punt & Branch Labs (U Washington)

Gavin Fay

02/03/2016

Lab schedule

1/27: Introduction to R and R Studio, working with data
2/03: Intro to plotting, manipulating data
2/10: Probability, linear modeling, PCA
2/17: Programming practices, conditional statements
2/24: Creating functions, debugging
3/02: Permutation analysis
3/09: Advanced plotting

Review

```
> weights <- c(2.3,5.4,7.5,9)
> weights #print a vector
[1] 2.3 5.4 7.5 9.0
> weights[c(1,3)] #vector elements 1 & 3
[1] 2.3 7.5
> mean(weights)  #mean of a vector
[1] 6.05
> weights[weights>=5 & weights<8] #subset dataframe
[1] 5.4 7.5
> hills$climb[2]  #element by vector name in dataframe
[1] 2500
> hills[which.max(hills$dist),] #row with max dist
            dist climb    time
Lairig Ghru   28  2100 192.667
> #races with climb > mean(climb)
> rownames(hills)[hills$climb>mean(hills$climb)]
 [1] "Carnethy"        "Ben Lomond"      "Goatfell"        "Be
 [5] "Lairig Ghru"     "Dollar"          "Lomonds"         "Ca
```

Recommended reading

An introduction to R (Venables et al.)
– `http://cran.r-project.org/doc/manuals/R-intro.pdf` -
Today's material: Chapters 4-7, 12.

Missing values (NA)

```
> weights <- c(25,34,75,NA,21,32,NA)
```

Many functions do not handle missing values by default.

```
> mean(weights)
[1] NA
> mean(weights,na.rm=TRUE)
[1] 37.4
```

Omit missing values.

```
> na.omit(weights)
[1] 25 34 75 21 32
attr(,"na.action")
[1] 4 7
attr(,"class")
[1] "omit"
```

Can also use is.na() to handle missing values.

```
> weights[!is.na(weights)]
[1] 25 34 75 21 32
```

Matrices and Arrays

Data frames: set of vectors of different types
Matrices: set of vectors of same type

```
> x <- matrix(data=1:6,nrow=2,ncol=3,byrow=FALSE)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Matrices and Arrays

Matrices can be formed by column and row binding

```
> row.mat <- cbind(1:3,4:6)
> row.mat
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> row.mat <- rbind(1:3,4:6)
> row.mat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Matrices and Arrays

`dim()` can be used as for dataframes to get dimensions of a matrix.
Matrices can also be created by specifying the dimensions.
Create 3+ dimensional arrays by adding dimensions to the `dim`.

```
> x <- array(data=1:6,dim=c(2,3))
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> x <- array(data=1:12,dim=c(2,3,2))
> x
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

Lists

The most flexible data structure.
Set of objects that are of varying length and mode.

```
> description <- "weights of fish (kg)"
> fish.list <- list(metadata=description,
+                   nfish=length(weights),data=weights)
> fish.list
$metadata
[1] "weights of fish (kg)"

$nfish
[1] 7

$data
[1] 25 34 75 NA 21 32 NA
> fish.list$nfish
[1] 7
> fish.list[[1]]  # [[]] extracts elements of lists
[1] "weights of fish (kg)"
> fish.list[[3]][1]
[1] 25
```

Categorical variables

Factors are vectors with discrete values assigned to each element.
R automatically specifies categorical variables as factors when
- creating data frames
- reading in data from files
- behind the scenes in many other cases

Can specify a categorical variable as a factor using `factor()`.

```
>  substrate <- c("cobble","mud","sand")
>  is.factor(substrate)
[1] FALSE
>  substrate.fac <- factor(substrate)
>  substrate.fac
[1] cobble mud    sand
Levels: cobble mud sand
>  is.factor(substrate.fac)
[1] TRUE
```

Numbers to factors

Categorical variables are often coded numerically.

```
>  substrate <- c(1,1,2,2,3,2,1,3)
>  substrate.fac <- factor(substrate,
+                         labels=c("cobble","mud","sand"))
>  substrate.fac
[1] cobble cobble mud    mud    sand   mud    cobble sand
Levels: cobble mud sand
```

To find the levels of a factor:

```
> levels(substrate.fac)
[1] "cobble" "mud"    "sand"
```

`droplevels(myfactor)` will remove unused levels.

Lab exercise 1/4

1. Create a 2x2 matrix `Amat` and a 2x3 matrix `Bmat`, each filled with unique numbers.
2. Combine `Amat` and `Bmat` into a 2x5 matrix `Cmat` and a 5x2 matrix `Dmat`.
3. Create a factor `xfactor` from the following vector such that 1 is immature and 2 is mature.
   `maturity <- c(1,1,2,1,2,2,2,1,1,1)`
4. Create a list called `data` that contains matrices `Amat`, `Bmat`, and `xfactor`.
5. Extract the first row of the matrix `Amat` from the list `data`.
6. Change to `NA` the value in row 1 and column 1 of matric `Bmat` within `data`.

The working directory

Getting R to know where files are is important when reading in data from file, or code from other R scripts.
R workspaces have a 'working directory', which can be printed using `getwd()` and changed using `setwd()`.

```
> getwd()
[1] "/home/gavin/Dropbox/Courses/sandbox"
> setwd('~/classes/advpopmod/labs/')
```

Projects in RStudio handle this in an efficient manner.
Hint: to overcome differences between Operating Systems in how directory structures are named, use `file.path()`

```
> dir <- 'classes/biostats2/labs' #Won't work in Windows.
> dir <- file.path('classes','biostats2','labs')
> dir
[1] "classes/biostats2/labs"
```

Reading in data

So far we have either typed in data values, or used built-in datasets.

3 common functions to read data from files.

1. `scan()`

- flexible, reads data into a vector.
- very fast, good for large or messy data.

2. `read.table`

- easy to use, reads data into a data frame.

3. `read.csv`

- special case of above, useful for comma-separated data.

scan()

```
> wts <- scan('weights.txt',n=100)
> wts[1:10]
 [1] 32 36 27 32 20 22 14 23 32 31
> summary(wts)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   11.0    21.0    26.0    25.7    30.0    46.0
```

Reading from a file with `read.table`

`read.table()` has many options (useful defaults listed below)

`header=T` first row has names for columns

`sep=" "` how are entries separated (white space)

`na.strings=NA` which values are treated as `NA`s

`skip=0` the number of lines to skip before reading in data

`nrows=-1` number of lines of data to read (-1 means all)

`col.names=c("a","b")` names for columns

Data from Finnish lake Laengelmavesi

Save `Laengelmavesi2.csv` to your computer.
Either to your project directory or create a directory called 'data'.

```
> data <- read.table(file="Laengelmavesi2.csv",
+                    header=TRUE,sep=",")
> # R will look for the file in the working directory.
> # Provide the directory path to the file if it is elsewhe
> head(data,n=3)
  species length weight height
1   Bream   25.4    242   38.4
2   Bream   26.3    290   40.0
3   Bream   26.5    340   39.8
```

Data can be downloaded from a website, but often better to have on your computer.
`data <- read.table(weblink.here,header=TRUE)`

Text fields in data files

`read.table` & `read.csv` treat text as factors.

Use argument `stringsAsFactors=FALSE` to suppress.

```
> data <- read.csv(file="Laengelmavesi2.csv",
+                  header=TRUE)
> unique(data$species)
[1] Bream     Whitefish Roach     Parkki    Smelt     Pike
Levels: Bream Parkki Perch Pike Roach Smelt Whitefish
> is.factor(data$species)
[1] TRUE
> data <- read.csv(file="Laengelmavesi2.csv",
+                  stringsAsFactors=FALSE,header=TRUE)
> is.factor(data$species)
[1] FALSE
```

Subsetting data

When NAs are involved, logical expressions may go awry.

subset() extracts portions of a data frame or matrix while handling NAs.

```
subset(object,logical expression, variable selection)
```

```
> subset(data,subset=weight>1500)
    species length weight height
100    Pike   60.0   1600   15.0
101    Pike   60.0   1550   15.0
102    Pike   63.4   1650   15.9
> subset(data,subset=weight>=1600,
+        select=c(species,length))
    species length
100    Pike   60.0
102    Pike   63.4
```

The `apply()` function

Very flexible function.

`apply(X, MARGIN, FUN, ...)`

- X = matrix
- MARGIN: 1=rpws, 2=columns
- FUN: an `R` function (can be user-defined, see later)

```
> M <- matrix(1:12,nrow=3)
> M
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> apply(X=M,MARGIN=2,FUN=mean)
[1]  2  5  8 11
> apply(X=M,MARGIN=1,FUN=mean)
[1] 5.5 6.5 7.5
```

for list operations, see `sapply()` and `lapply()`

rowMeans and colMeans

Easy way to get the means of rows or columns.

```
> M <- matrix(1:12,nrow=3)
> colMeans(M)
[1]  2  5  8 11
> rowMeans(M)
[1] 5.5 6.5 7.5
```

tapply()

Apply a function to a vector using a categorical variable.

```
> lengths <- sample(1:100,size=20,replace=TRUE)
> lengths
 [1]  73  21  55  83  54  34  19  22 100  26  34   6  66  3
[18]  76   6  70
> maturity <- sample(c("mature","immature","unknown"),
+                    size=20,replace=TRUE)
> maturity
 [1] "mature"   "immature" "mature"   "immature" "mature"
 [7] "unknown"  "immature" "mature"   "immature" "mature"
[13] "unknown"  "immature" "immature" "mature"   "mature"
[19] "mature"   "unknown"
> tapply(X=lengths, INDEX=maturity, FUN=mean)
immature   mature  unknown
34.00000 48.87500 51.66667
```

Lab exercise 2/4 (Laengelmavesi)

Read in the data file `Laengelmavesi2.csv` and:

1. Display the number of observations for each species of fish.
2. Find the overall mean lengths, weights, and heights of fish in the data.
3. Find the range of the lengths of Perch.
4. Find the mean length of fish with weight greater than 1000g.
5. Calculate the mean and CV of lengths and weights for each species.
6. Create a new data frame that just contains the Pike data.
7. *bonus* With the Pike data, create a new factor for small and large based on the weights.

Plotting in R

Recommended reading
An introduction to R (Venables et al.), Chapter 12
`http://cran.r-project.org/doc/manuals/R-intro.pdf`

`R` graphics 2nd Edition (Paul Murrell, 2011)
Chapters 1 and 2
Pdf of the 1st edition here:
`https://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html`
`R` code available for all plots in 2nd edition:
`https://www.stat.auckland.ac.nz/~paul/RG2e/`

Graphics options in R

Three main options for plotting

- base graphics (easy to change, highly modifiable)
- `lattice` (multipanel plots, not used much now due to)
- `ggplot` (great for quick multipanel plots, not as easy to change defaults)

Dealing with base plotting here.

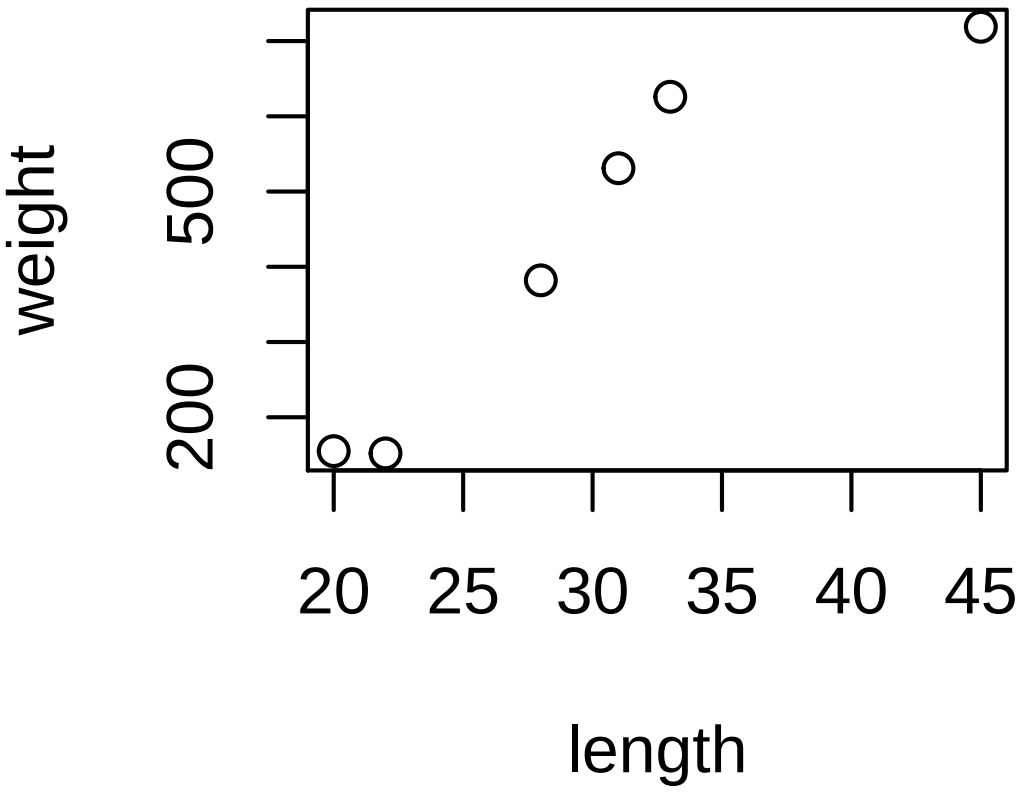
Base graphics: plot()

`plot()` is the generic function for plotting `R` objects. `Laengelmavesi.csv` contains mean weights and lengths for six species of fish.

```
> fish <- read.csv(file="Laengelmavesi.csv",header=TRUE)
> fish
    species length weight height
1     Bream     33    626   15.2
2 Whitefish     31    531   10.0
3     Roach     22    152    6.7
4    Parkki     20    155    9.0
5      Pike     45    719    7.7
6     Perch     28    382    7.9
```

Using the `plot()` command

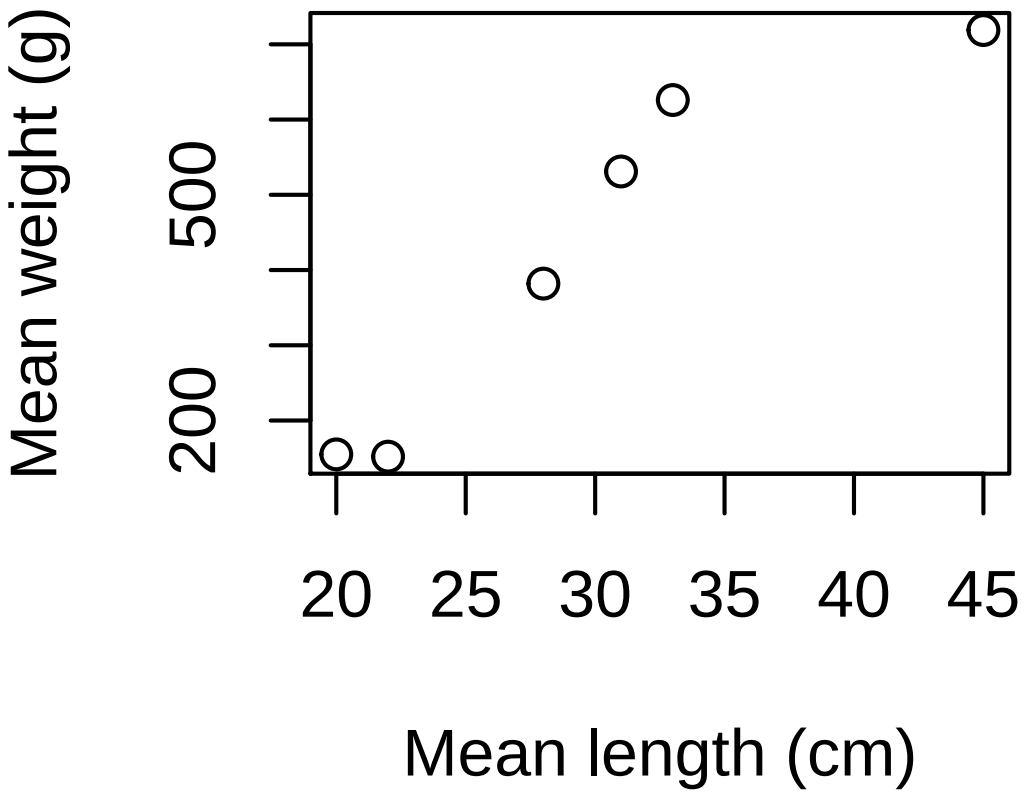
```
> plot(x=fish$length, y=fish$weight)
> plot(weight~length, data=fish)
```



Axis labels

By default R uses variable names as axis labels.
Use `xlab` and `ylab` to change the labels.

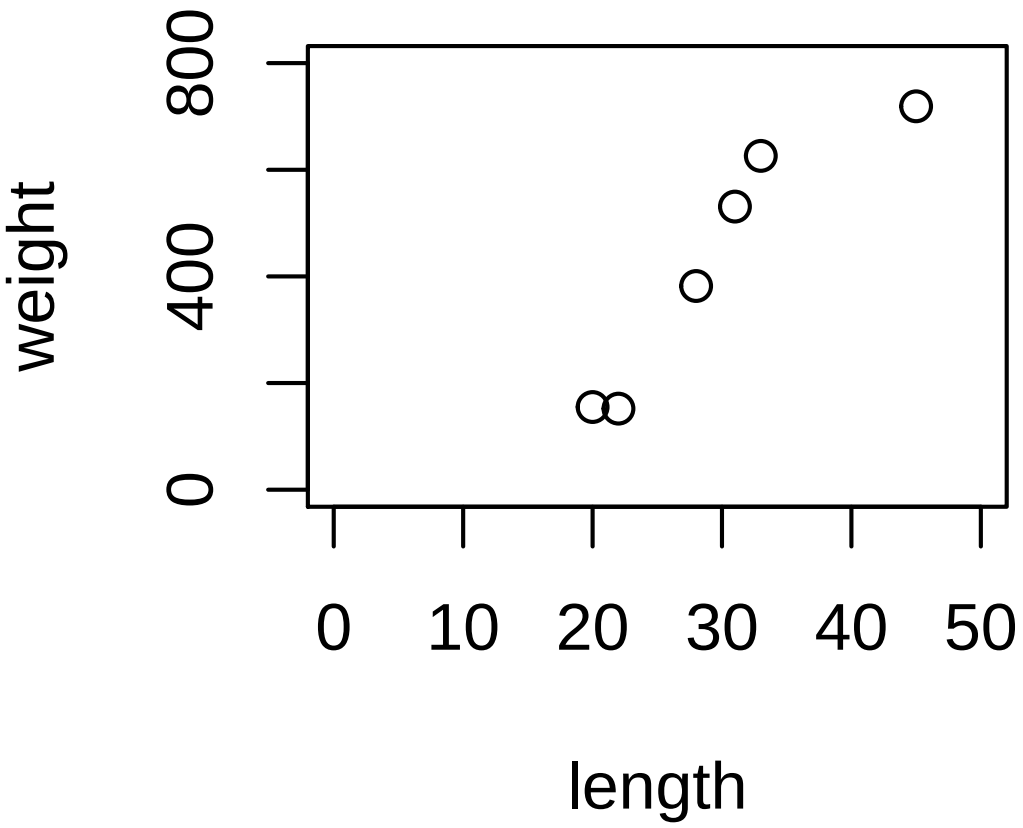
```
> plot(weight~length,data=fish,xlab="Mean length (cm)",
+      ylab="Mean weight (g)")
```



Axis limits

R chooses x and y limits just larger than the range of the data. To change the default x and y values use `xlim` and `ylim`.

```
> plot(weight~length,data=fish,xlim=c(0,50),
+      ylim=c(0,800))
```



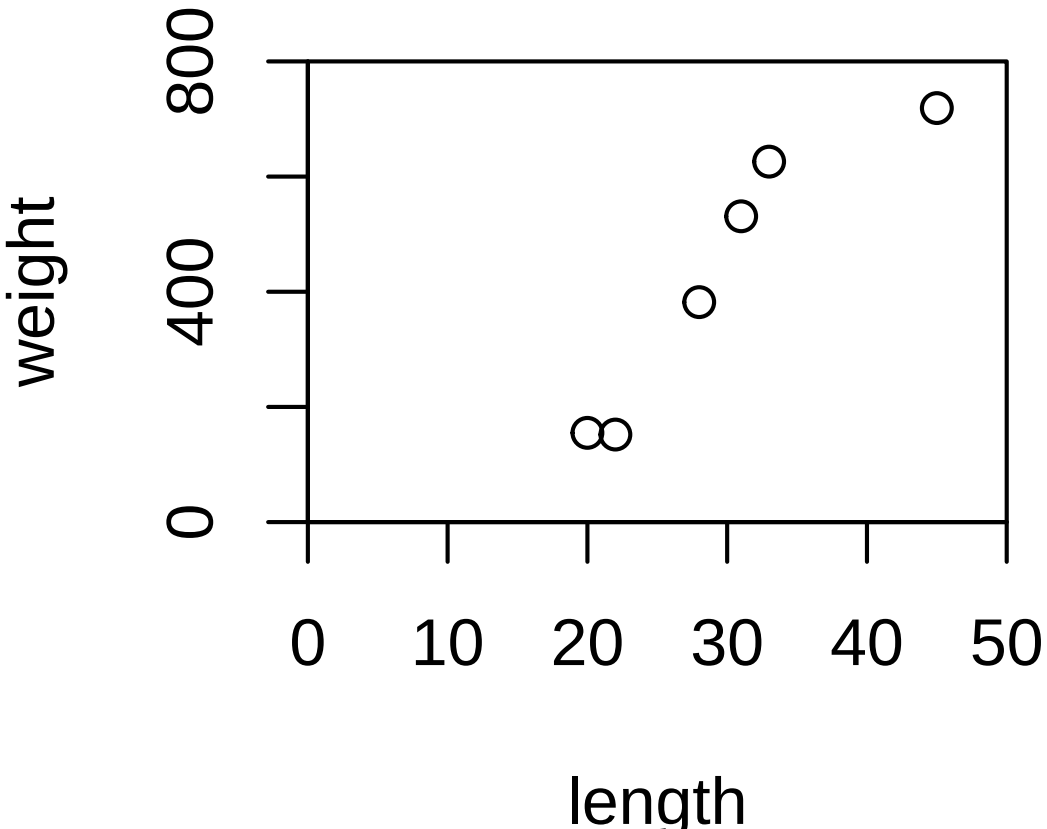
Remove spaces around zeroes

R adds space between the axis and 0.

This makes true zeros look like they are non-zeros.

Remove using `xaxs="i"` and `yaxs="i"` with `xlim` and `ylim`.

```
> plot(weight~length,data=fish,xlim=c(0,50),ylim=c(0,800),
+      xaxs="i",yaxs="i")
```



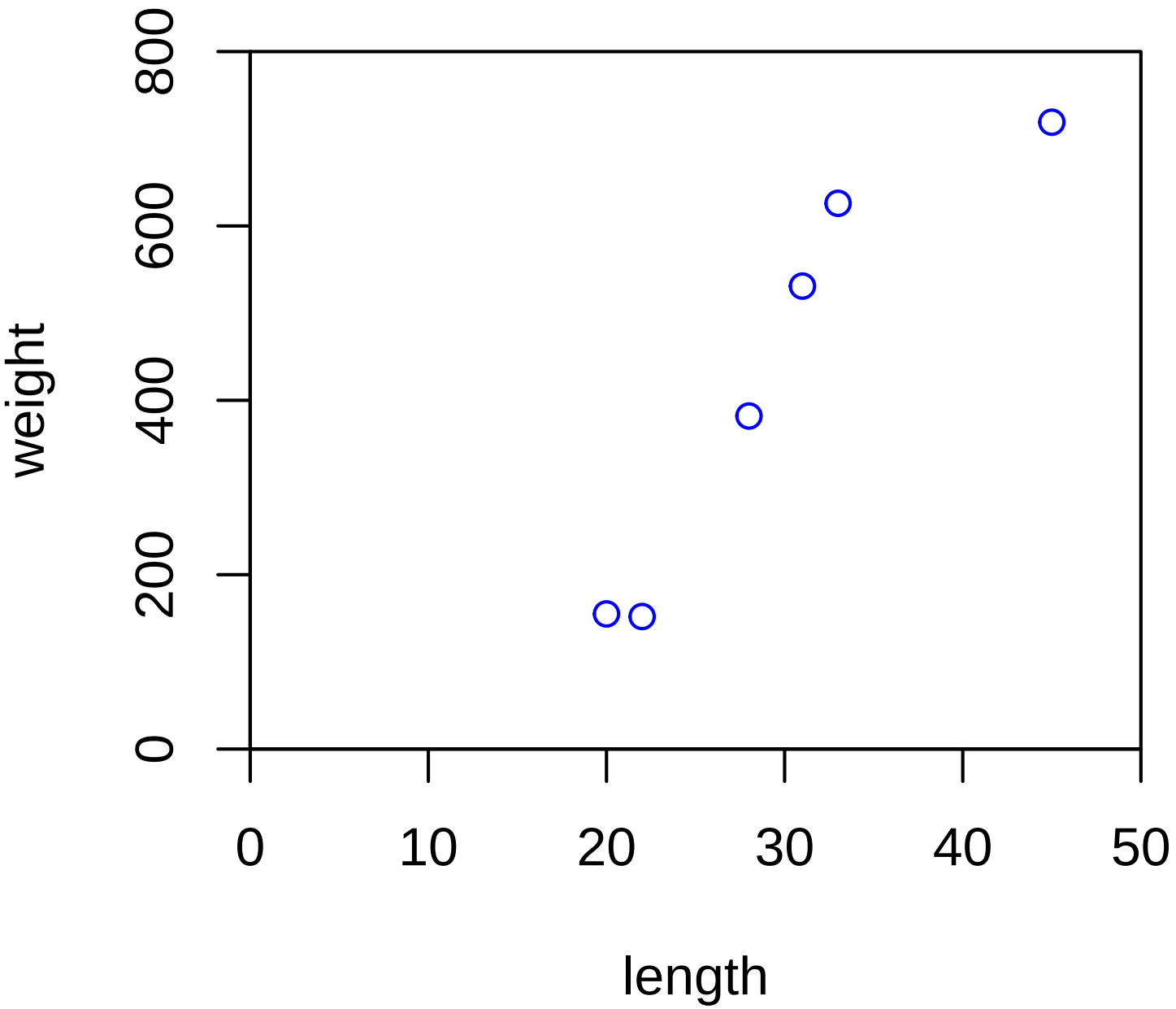
Colors

Colors of points, lines, text etc. can all be specified.

- `col` (default color)
- `col.axis` (tick mark labels)
- `col.lab` (x label and y label)
- `col.main` (title of the plot)

Colors can be specified as numbers or text strings
`col=1` or `col="red"`

```
>  plot(weight~length,data=fish,xlim=c(0,50),
+       ylim=c(0,800),xaxs="i",yaxs="i",col="blue")
```


weight
0
200
400
600
800
0
10
20
30
40
50
length

In-class exercise

Copy and paste the following command into your R script:

```
plot(weight~length,data=fish,xlim=c(0,50),
ylim=c(0,800),xaxs="i",yaxs="i",col="blue")
```

Experiment with different color names: col="red"

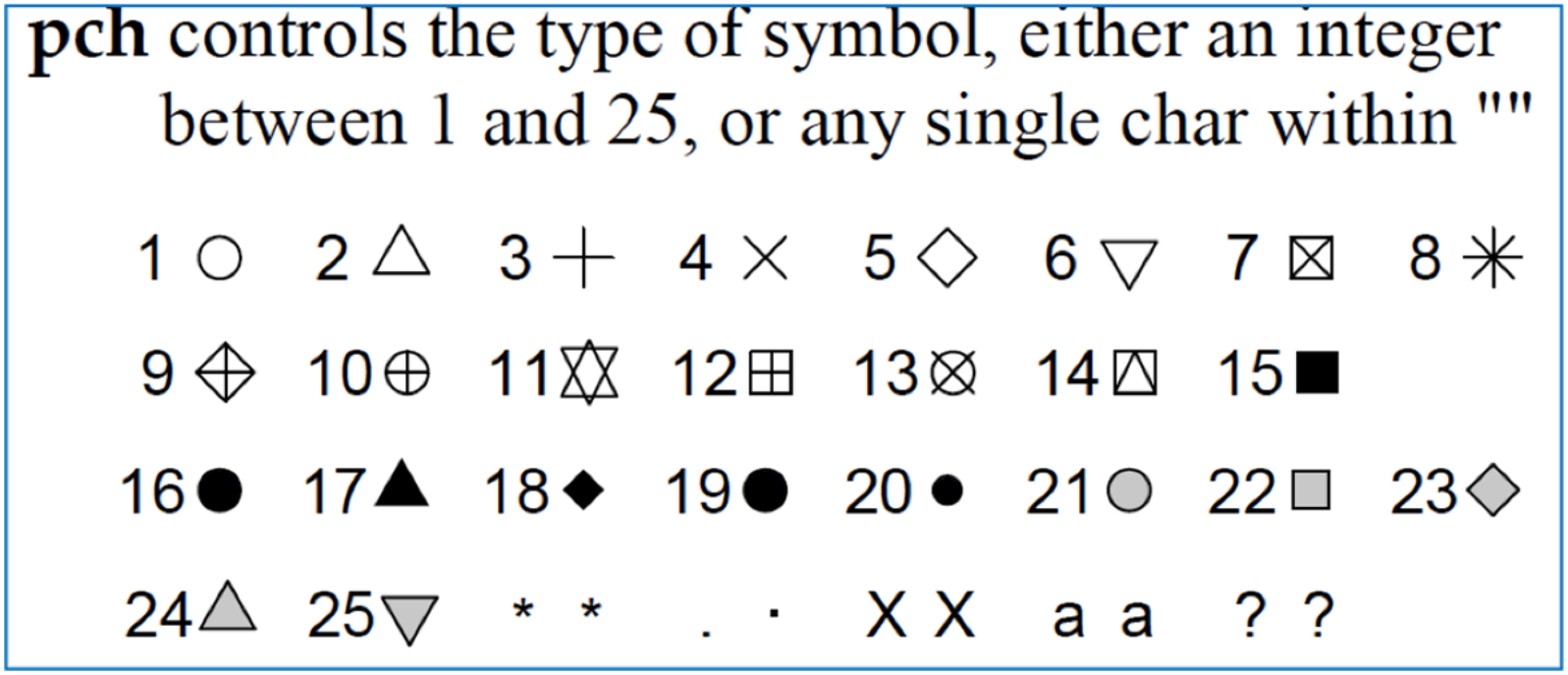
Try different color numbers: col=1, col=2

Try a vector of color numbers: col=c(2,4)

Experiment with changing the values for cex and pch

Plotting characters

The default plotting character is an open circle (`pch=1`) of size (`cex=1`).



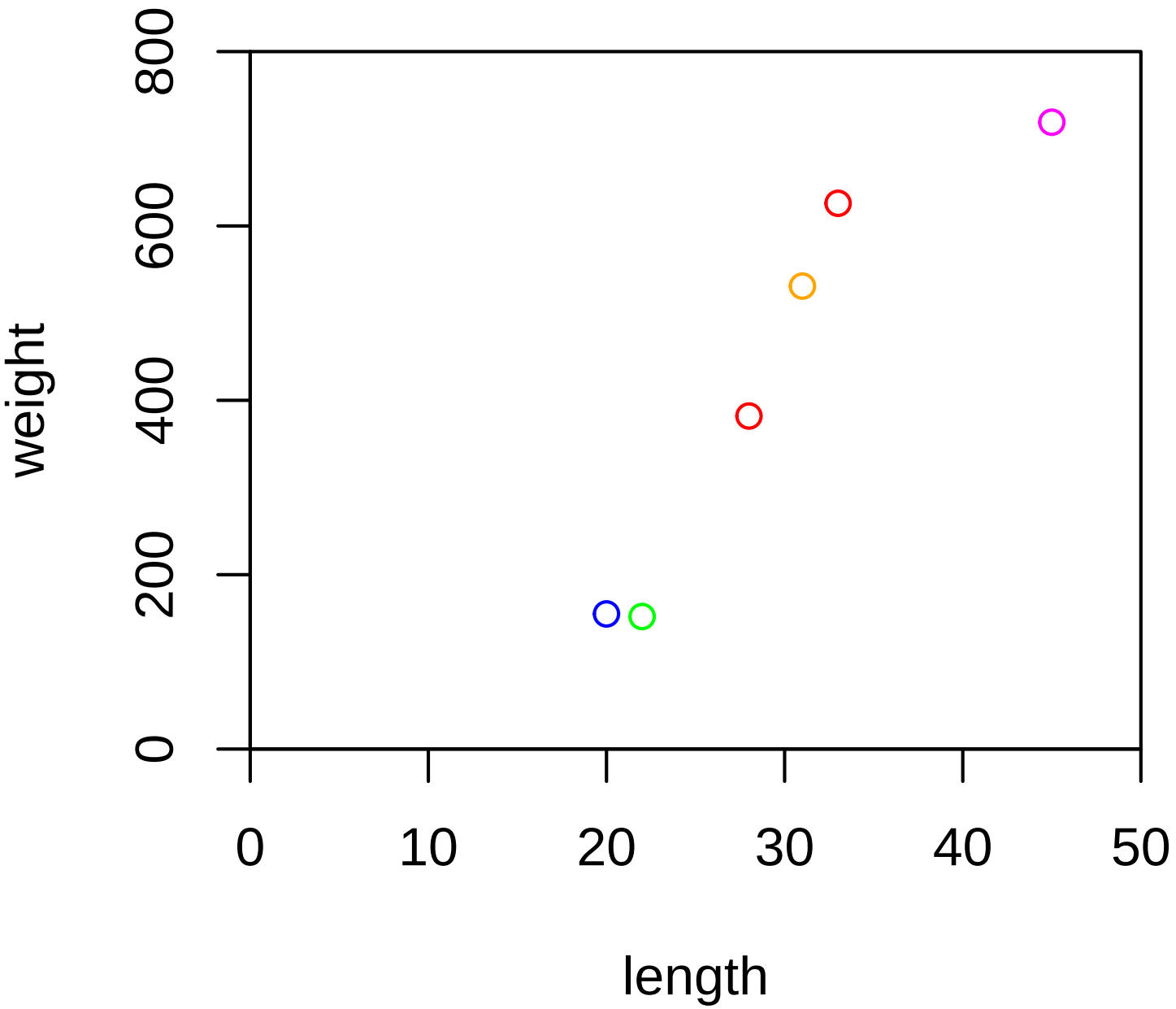
(R reference card 2.0)

Naming and finding Colo(u)rs

R has 657 **named** colors.

Color resources: - R color chart
http://research.stowers-institute.org/efg/R/Color/Chart/ColorChart.pdf
- ColorBrewer
http://colorbrewer2.org/

```
> colors()
> point.colors <- c("red", "orange", "green", "blue",
+           "magenta","black")
> plot(weight~length,data=fish,xlim=c(0,50),
+      ylim=c(0,800),xaxs="i",yaxs="i",col=point.colors)
```


weight
0
200
400
600
800
0
10
20
30
40
50
length

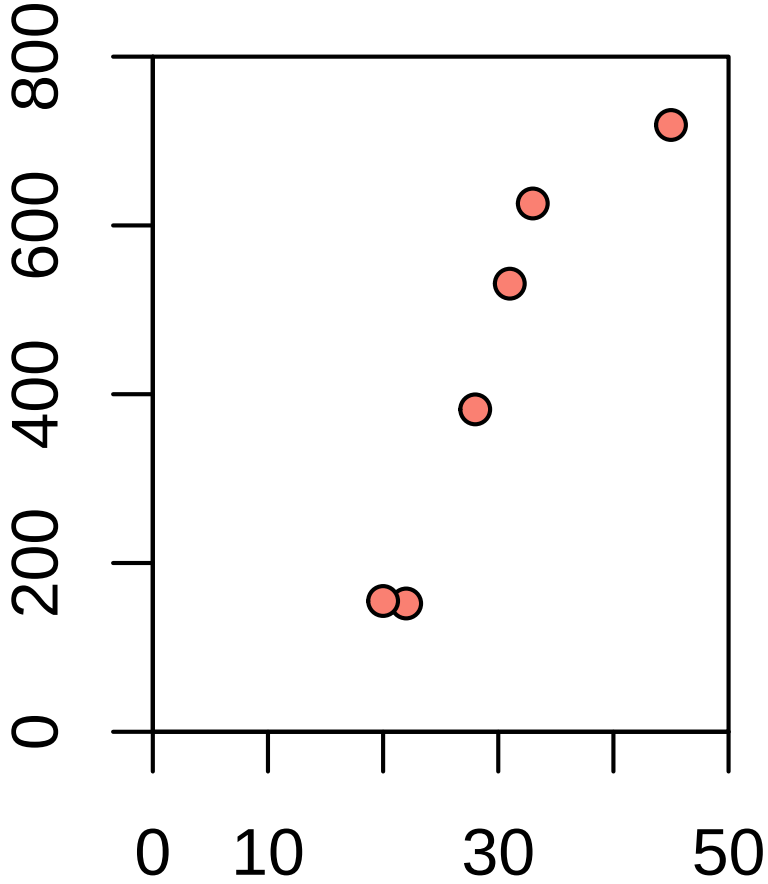
Useful options for points

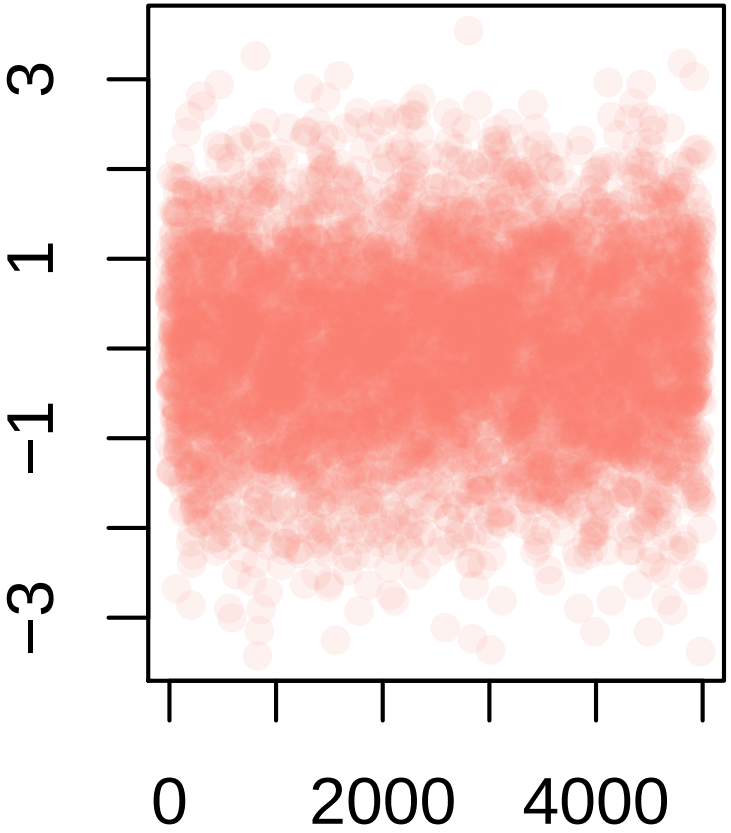
Use `pch=21` for filled circles, for example:
- Specify circle color with `col`
- Specify fill color with `bg`

```
> plot(weight~length,data=fish,xlim=c(0,50),
+      ylim=c(0,800),xaxs="i",yaxs="i",pch=21,
+      col="black", bg="salmon")
```

Transparent points are useful for large datasets.

```
>  col=rgb(t(col2rgb("salmon"))/255,alpha=0.1)
>  col=alpha("salmon",0.1) # in `scales` package
>  plot(rnorm(5000),col=alpha("salmon",0.1),pch=16)
```


0
200
400
600
800
0
10
30
50


−3
−1
1
3
0
2000
4000

Find ALL the plot parameters! `par()`

Only a few commands are listed in the `?plot` help.

There are LOTS of extra commands listed under `?par`.

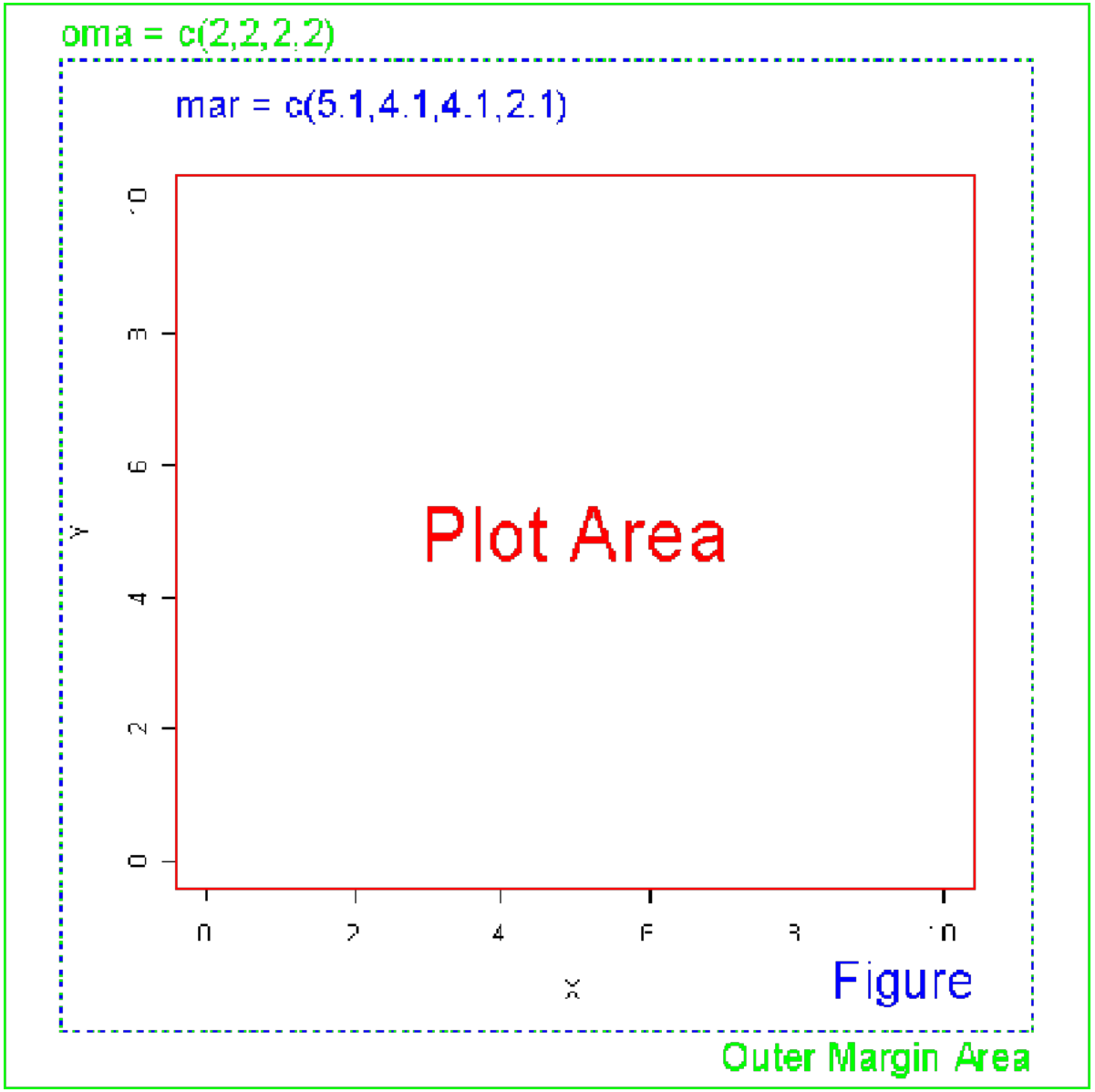
These can be added to **all** plotting commands.

Using `par()` by itself applies commands to multiple graphs.

Commands I find useful:
- `tcl` (tick length, use `tck` for gridlines)
- `yaxp` & `xaxp` (tick mark labeling)
- `mgp` (position of axis labels and title)
- `las` (style of text - parallel/perpendicular to axis)

External calls to `par`
- control margin size (`oma` & `mar`)
- multipanel plot layout (`mfrow` & `mfcol`)

Plot margins



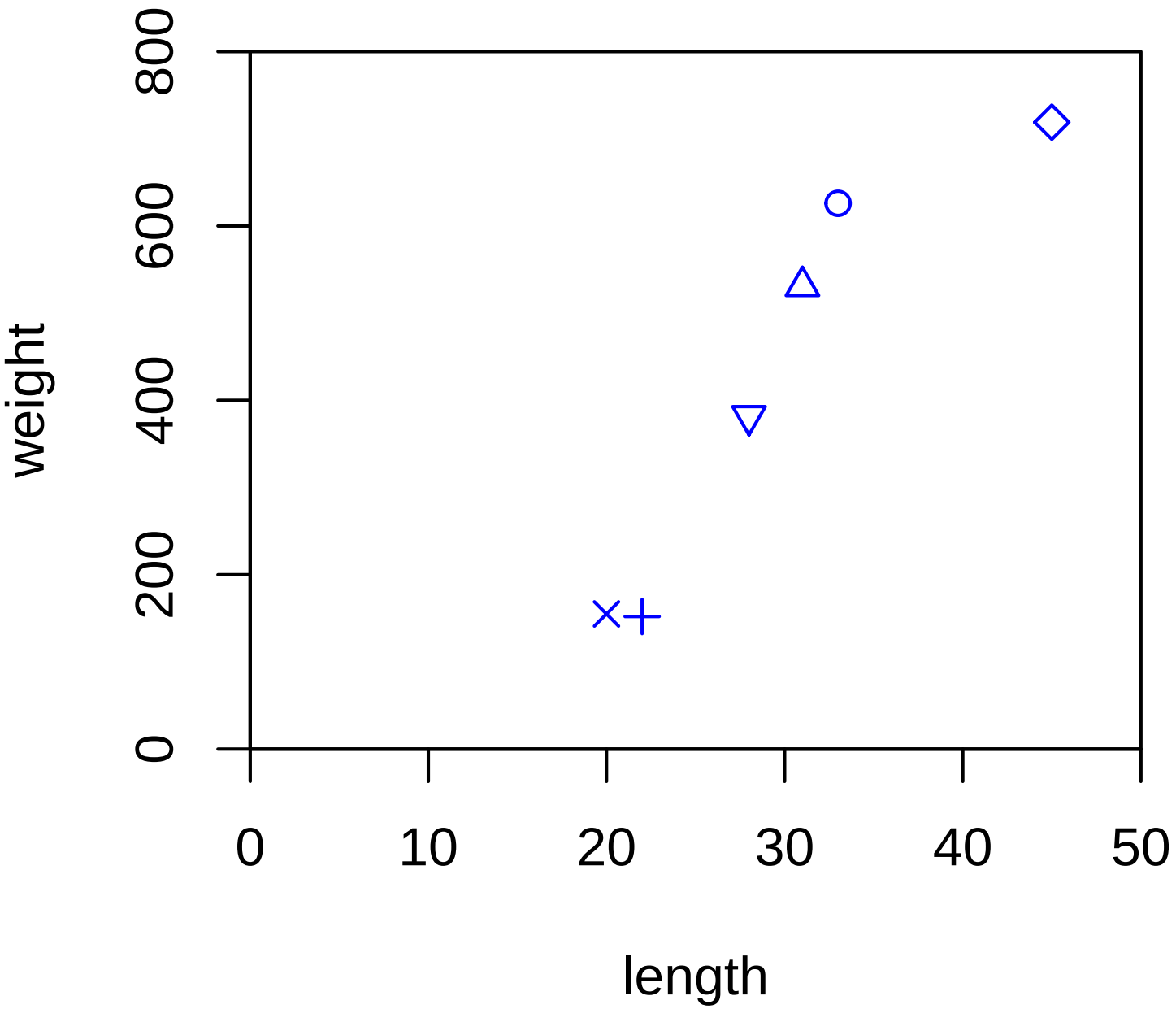
Vector options for plotting

Many plotting options can handle vectors.

Vectors are recycled if supply too few numbers.

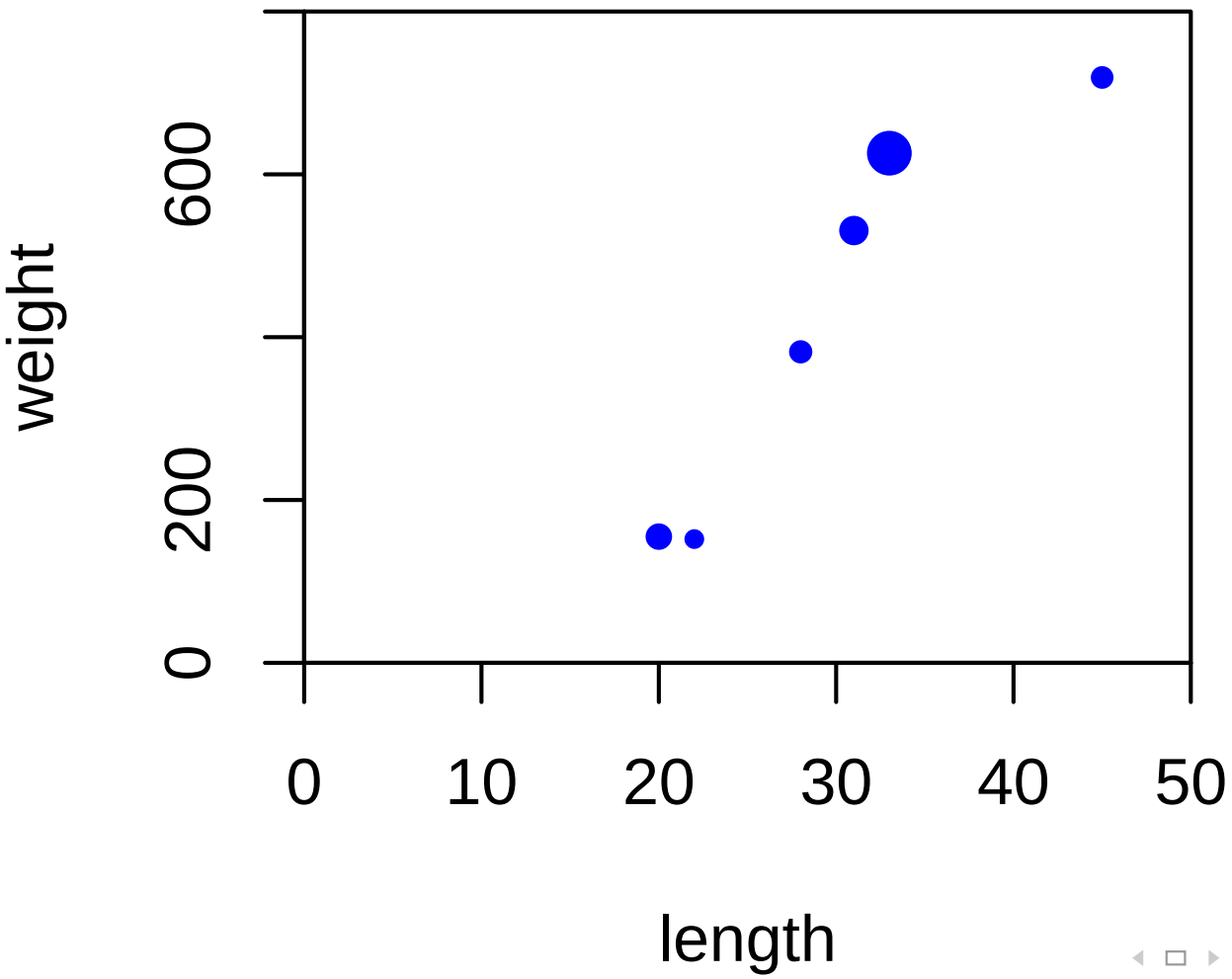
e.g. different point characters:

```
>  plot(weight~length,data=fish,xlim=c(0,50),
+       ylim=c(0,800),xaxs="i",yaxs="i",col="blue",
+       pch=1:6)
```


weight
0
200
400
600
800
0
10
20
30
40
50
length

Circle size proportional to body height

```
>  plot(weight~length,data=fish,xlim=c(0,50),
+       ylim=c(0,800),xaxs="i",yaxs="i",col="blue",
+       pch=16,cex=0.1*height)
```

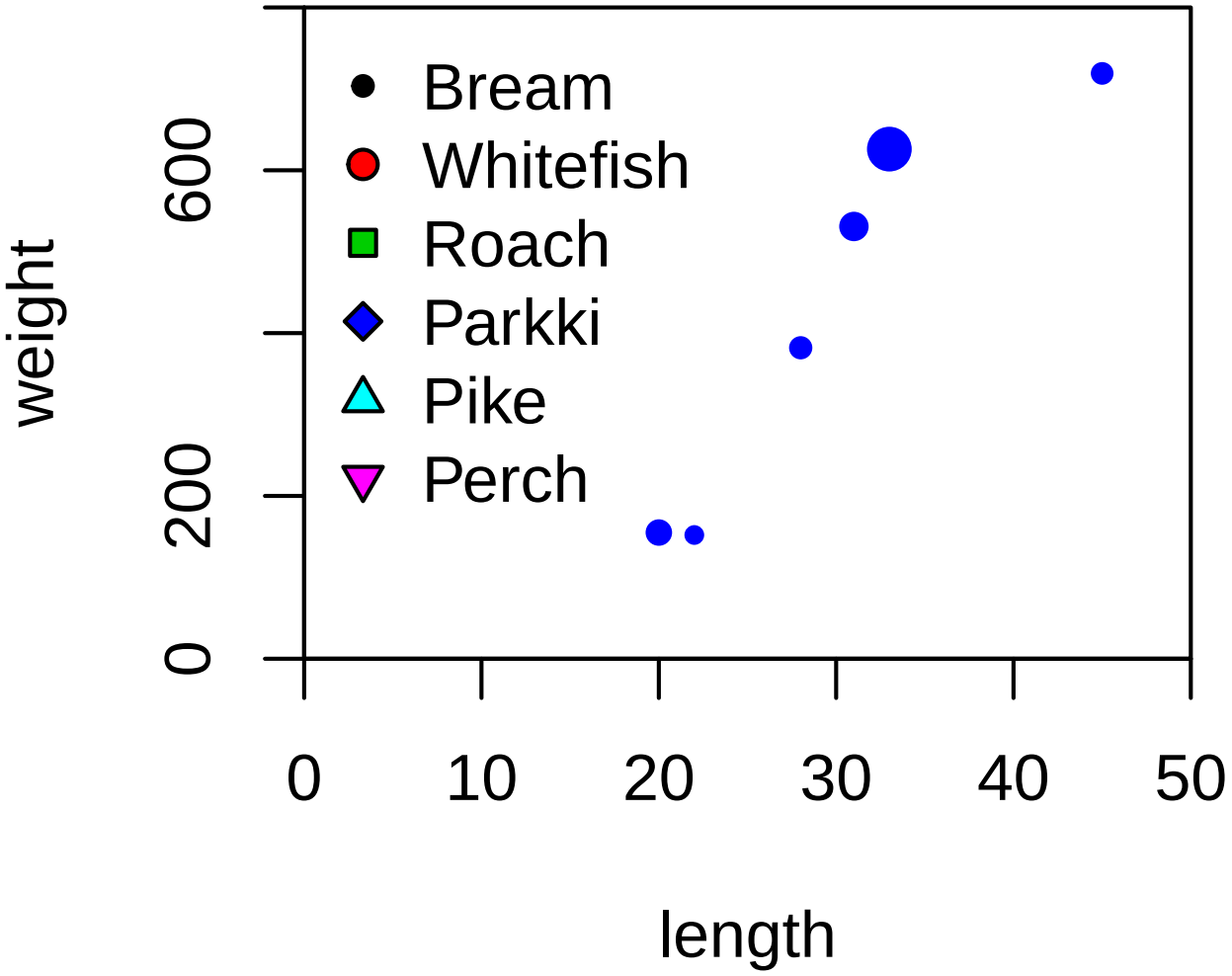


Adding legends

Look up help on `legend` function (`?legend`), many options in `par()` can be used here.

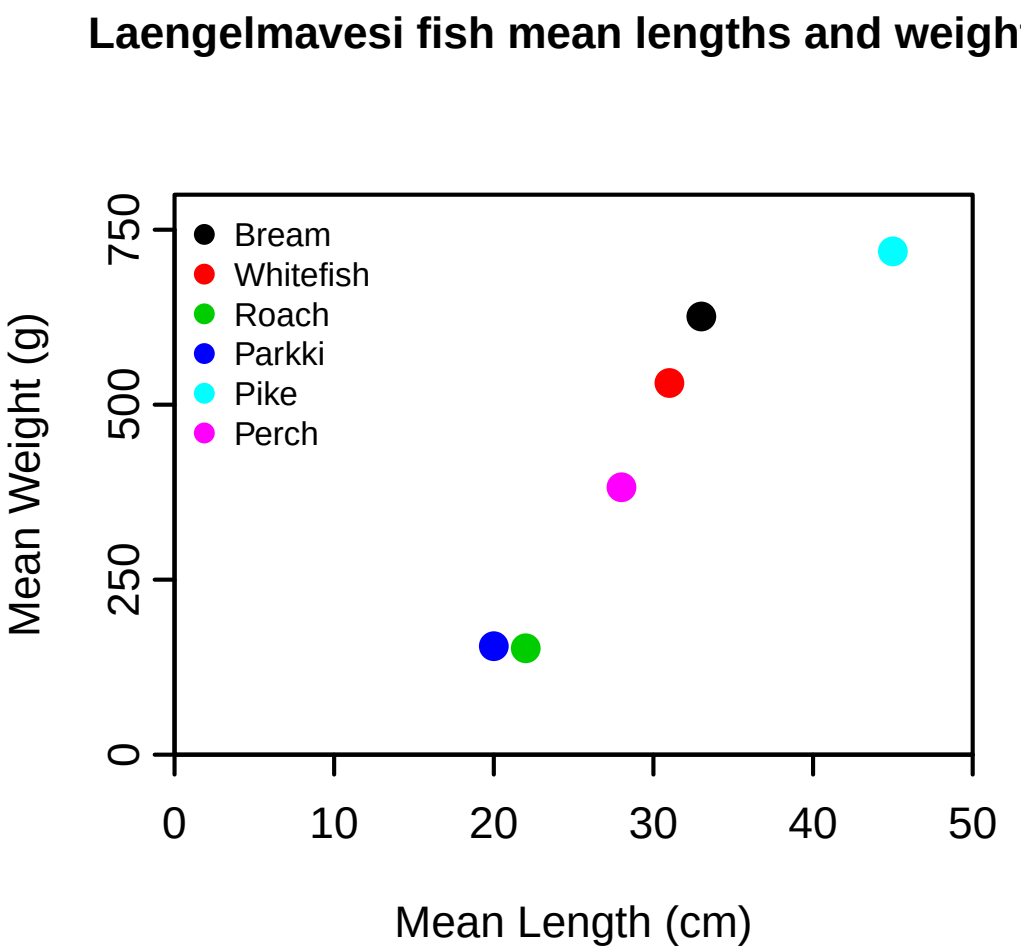
```
> plot(weight~length,data=fish)
>
> legend(x="topleft",   #position in plot, also x=0, y=600
+        legend=fish$species,  #vector of text strings
+        pt.bg=1:6,   # background color of points
+        pch=20:25,   # vector of symbol type
+        bty="n")     # no box around legend
```

If you want the legend to correspond to the plot, you need to specify identical symbols, sizes, and colors for the plot and the legend.



Lab exercise 3/4

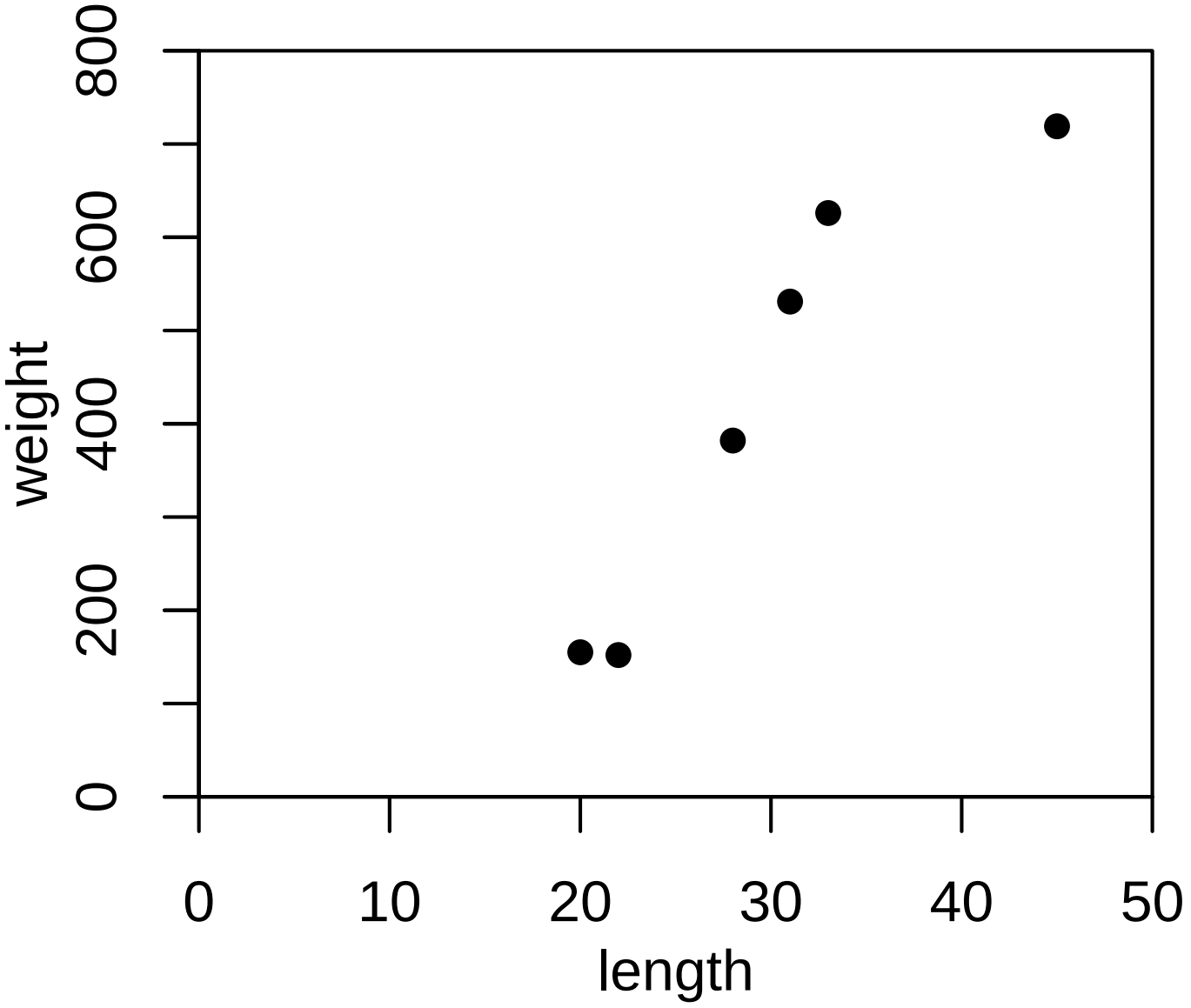
- Try to replicate as close as possible this graph.
- Colors are 1:6. Figure out how to add a title.



Advanced axis properties, `axis()`

For more control over axes, use the `axis()` function.
Suppress the x or y axis when creating the plot using `xaxt="n"` and `"yaxt="n"`.
Then add axes to whichever side they are needed.
`mtext()` can be used to add text to the margins.

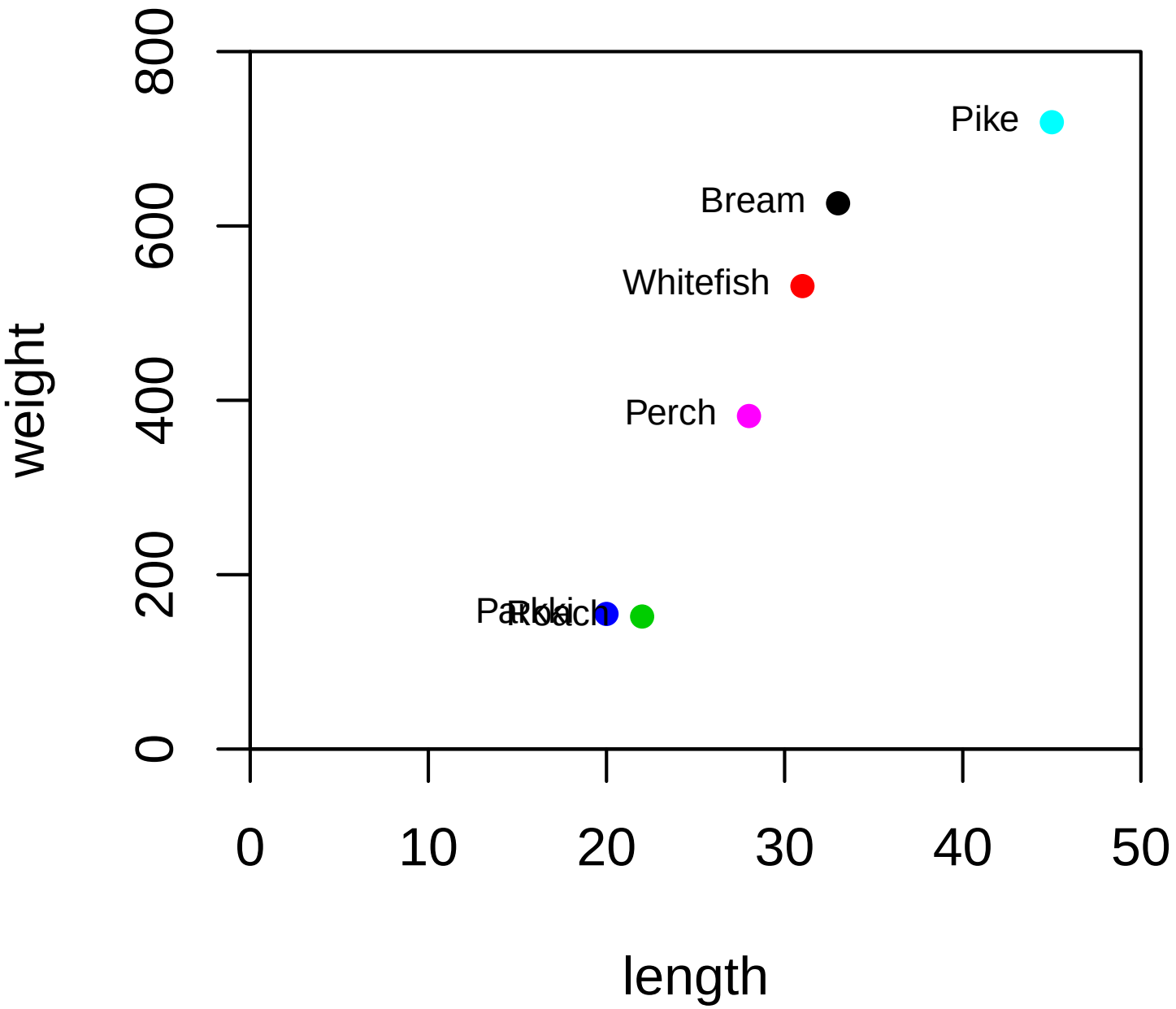
```
> plot(weight~length,data=fish,xlim=c(0,50),
+       ylim=c(0,800),xaxs="i",yaxs="i",
+       pch=16,xaxt="n",yaxt="n",xlab="",ylab="")
> axis(side=1, at=seq(0,50,10),labels=seq(0,50,10))
> axis(side=2, at=seq(0,800,100),labels=FALSE)
> axis(side=2, at=seq(0,800,200),labels=seq(0,800,200))
> mtext("length",side=1)
> mtext("weight",side=2)
```


weight
0
200
400
600
800
0
10
20
30
40
50
length

Labeling points using `text()`

`text()` allows you to add text to a plot, and it can take vector input.
After creating the plot, call `text()`

```
> plot(weight~length,data=fish,xlim=c(0,50),
+       ylim=c(0,800),xaxs="i",yaxs="i",
+       pch=16,col=1:6)
> text(x=fish$length,y=fish$weight,  #coordinates of text
+      labels=fish$species,  # text to insert at points ab
+      pos=2) #position of text relative to coordinates
>             #1= below, 2= left, 3=above, 4= right
```


Pike
Bream
Whitefish
Perch
Parkki
Roach
weight
0
200
400
600
800
0
10
20
30
40
50
length

Plot types

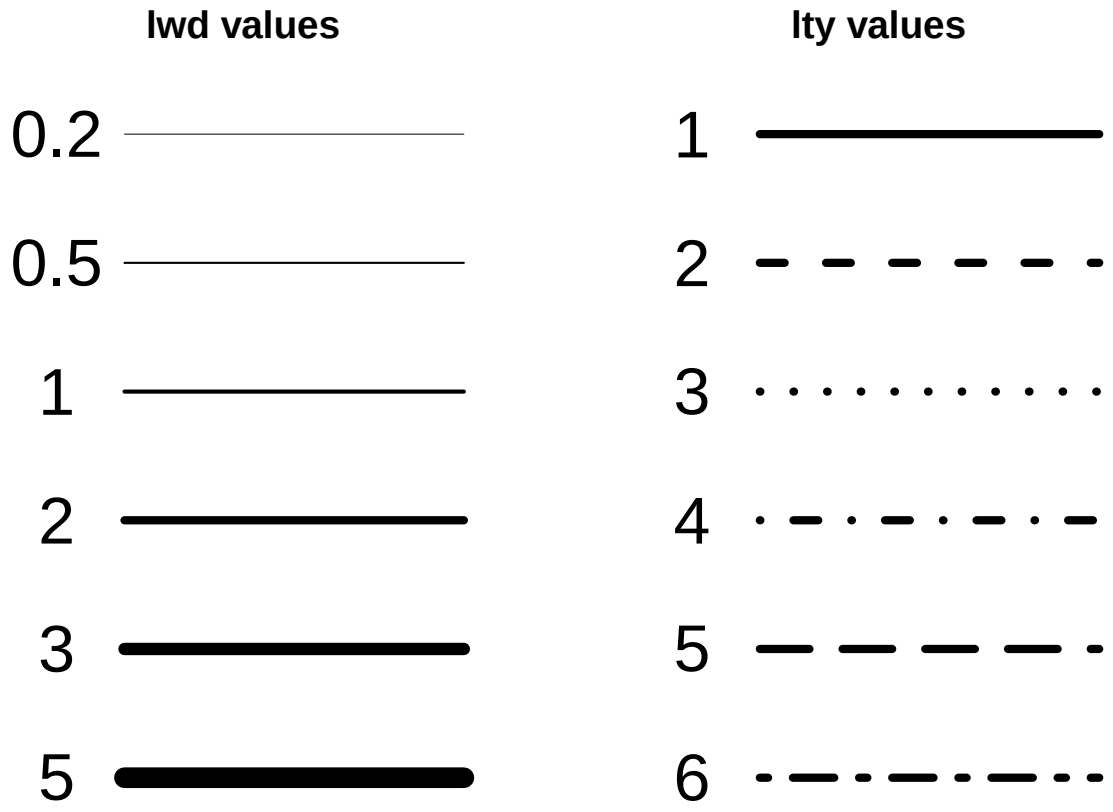
In the `plot()` command, `type` specifies the type of plot to be drawn.

- “p” points (default)
- “l” lines
- “b” both lines and points
- “c” lines part alone of “b”
- “o” overplotted
- “h” histogram-like vertical lines
- “s” stair steps
- “n” for no plotting

Line type and weight

`?lines` gives values for:

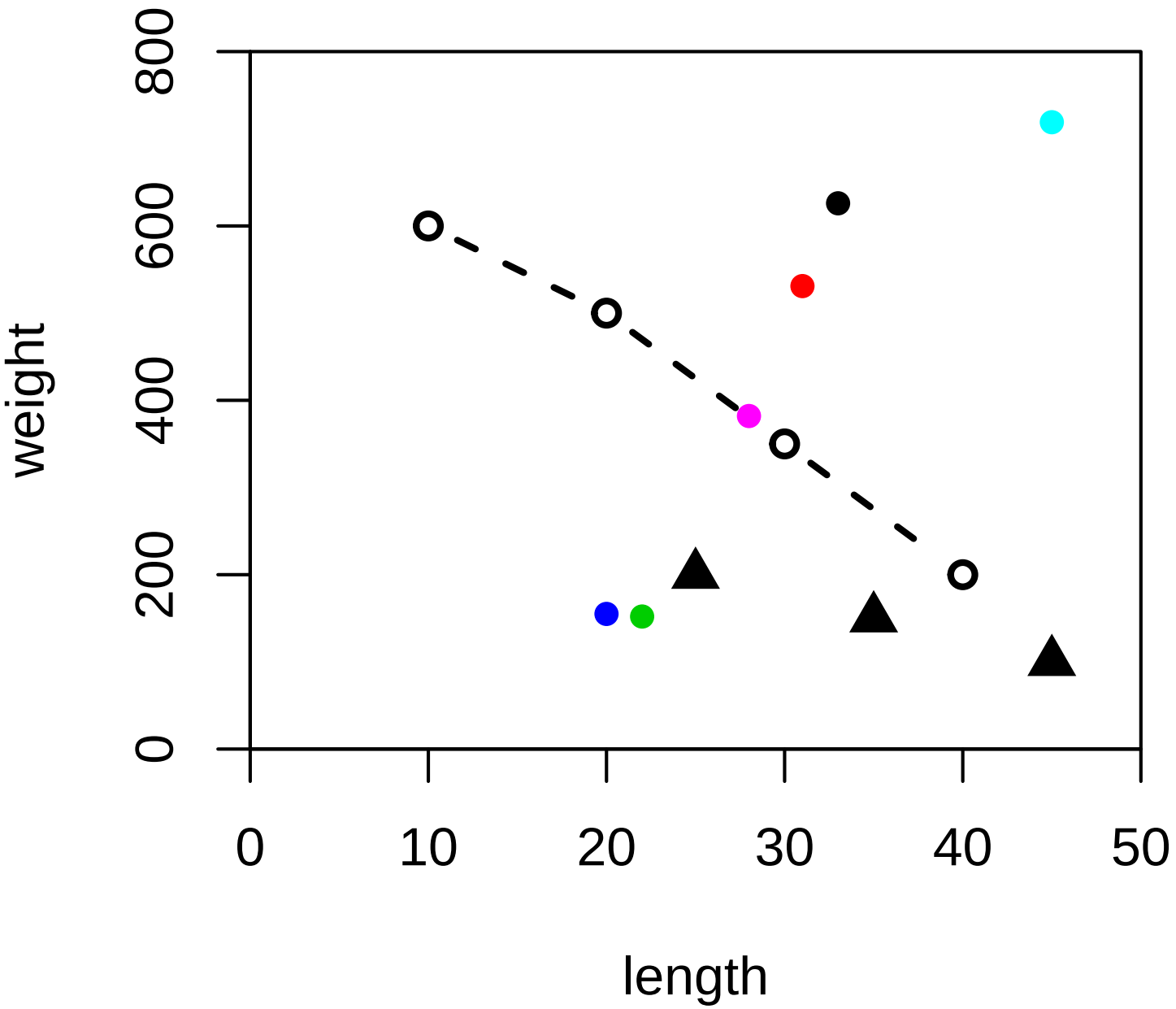
- `lty`, the line types
- `lwd`, the line widths



Adding points and lines

You can add a series of points or lines to the current plot using `points()` and `lines()`.

```
> plot(weight~length,data=fish,xlim=c(0,50),
+       ylim=c(0,800),xaxs="i",yaxs="i",
+       pch=16,col=1:6)
> lines(x=c(10,20,30,40),y=c(600,500,350,200),
+       lty=2,lwd=2,type="b")
> points(x=c(25,35,45),y=c(200,150,100),
+        cex=1.5,pch=17)
```


weight
0
200
400
600
800
0
10
20
30
40
50
length

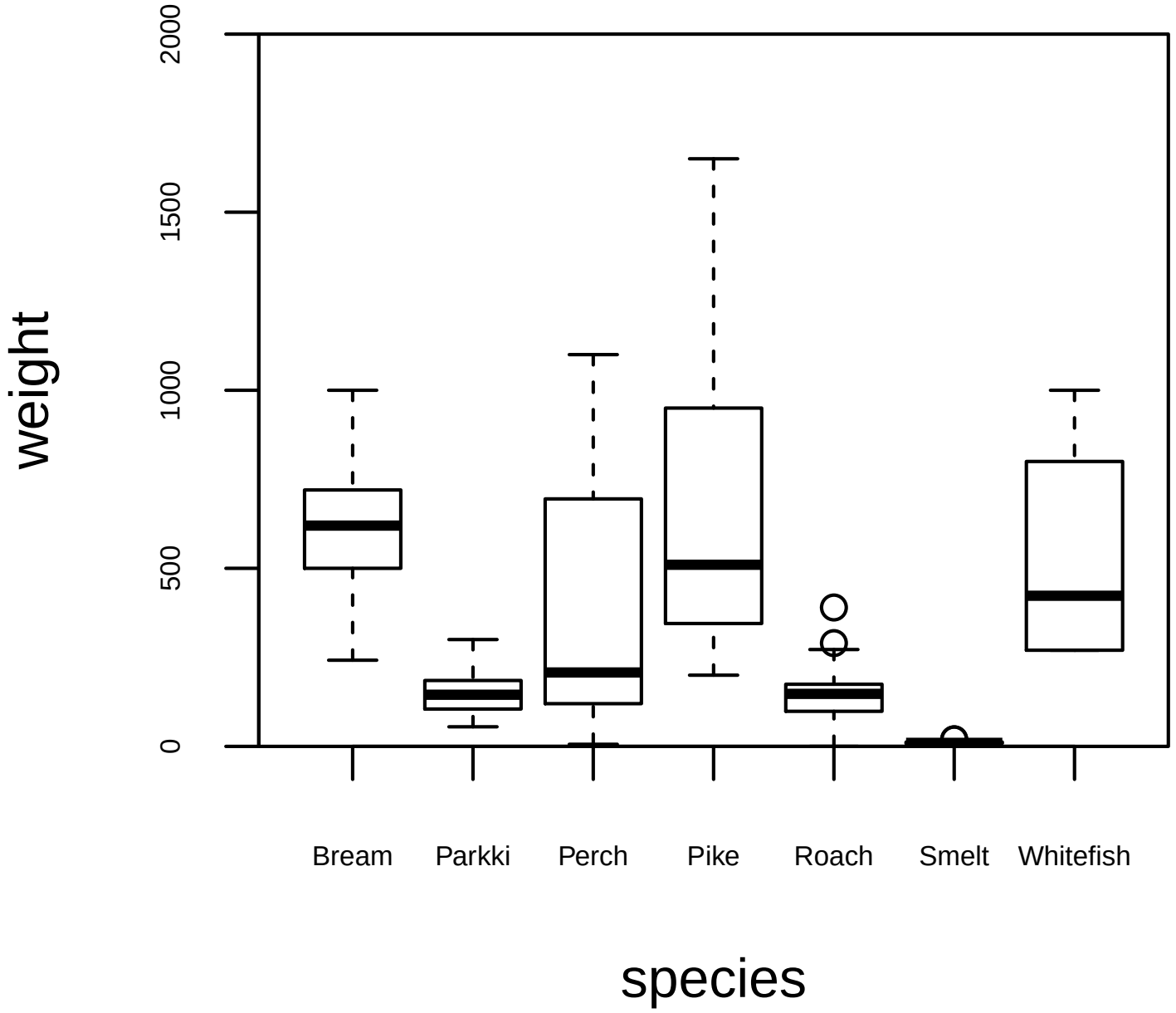
Other types of plots

- `plot()` is an overloaded function.
- what it returns depends on the type of objects that are given to it.
- there are versions of `plot` that provide useful outputs for many `R` functions.
- e.g. `plot.lm()` is a version that plots typical diagnostics from a linear model object. (but you still just type `plot(myobject)`).

Boxplots

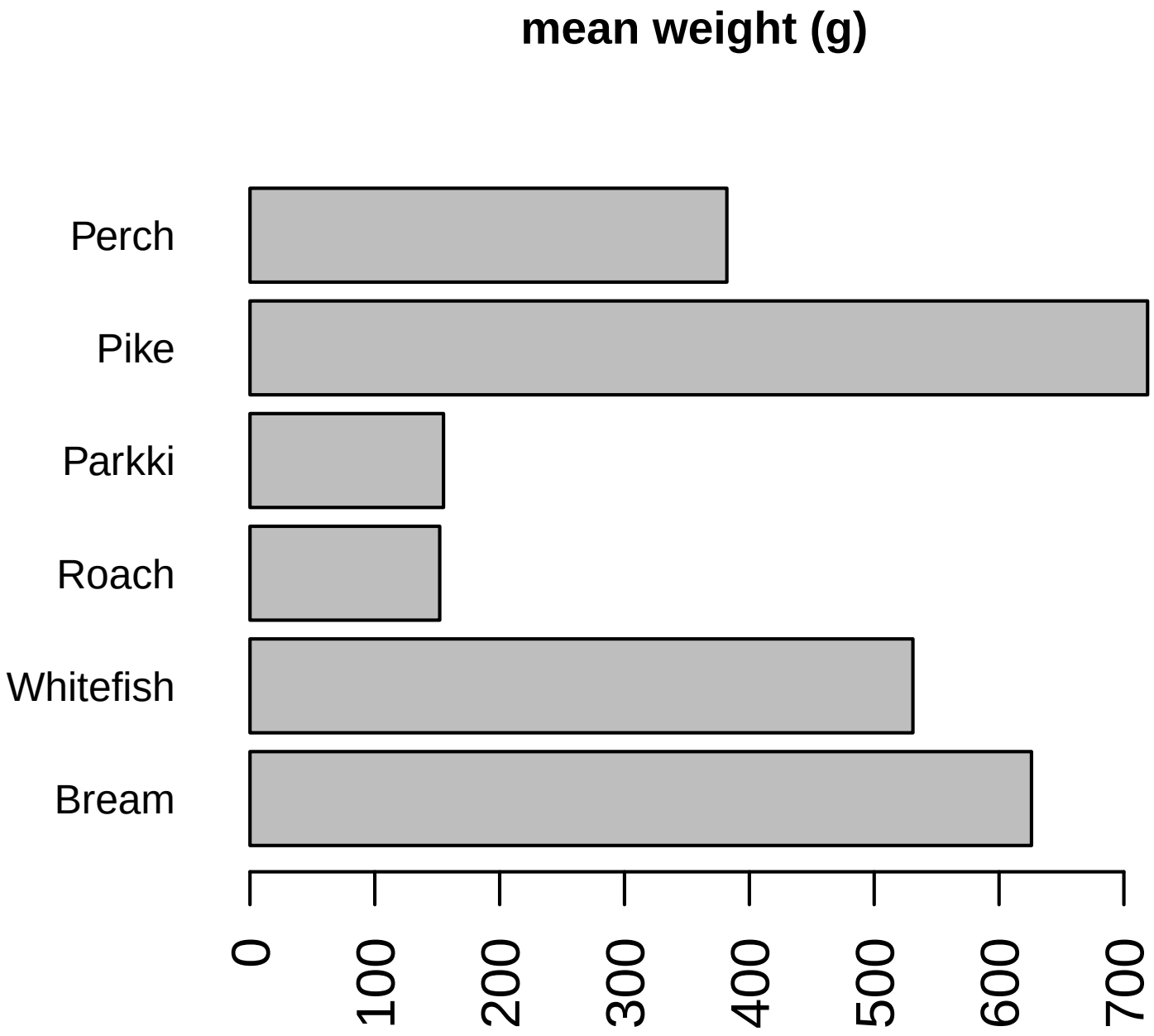
If the x variable in a typical call to `plot` is a categorical variable (factor), then the default plot is a boxplot.

```
> mydata <- read.csv(file="Laengelmavesi2.csv",
+                    header=TRUE)
> is.factor(mydata$species)
[1] TRUE
> plot(weight~species,data=mydata)
```



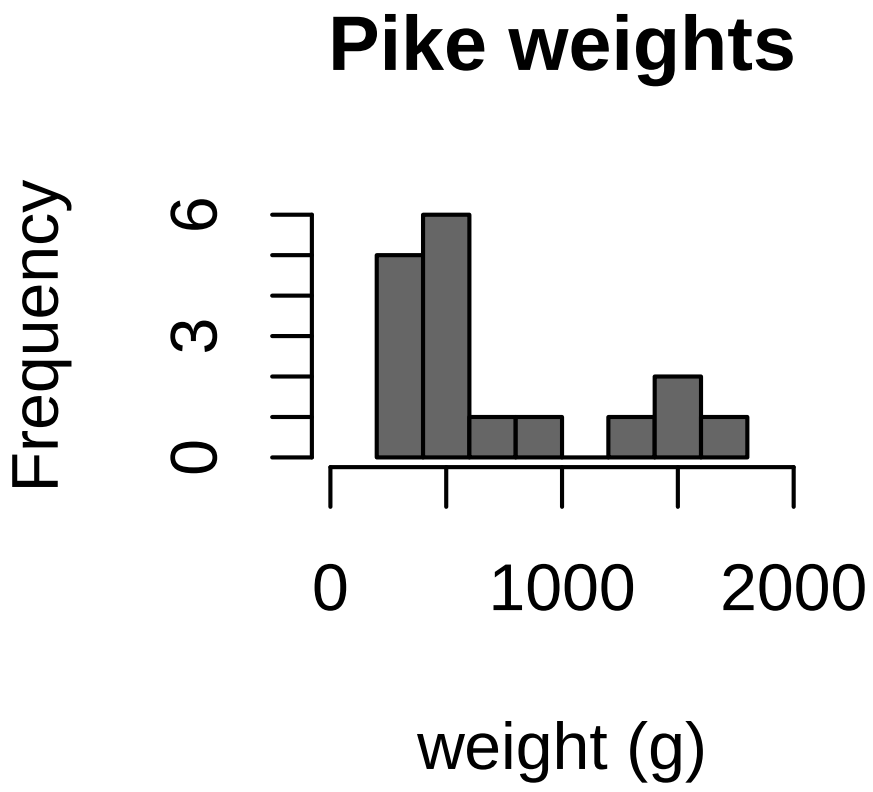
Alternatively, boxplots can be created using `boxplot()`

Bar plots, `barplot()`



Histograms, `hist()`

```
> mydata <- read.csv(file="Laengelmavesi2.csv",
+                    header=TRUE)
> hist(subset(mydata$weight,mydata$species=="Pike"),
+      xlim=c(0,2000),xlab="weight (g)",
+      col=gray(0.4),main="Pike weights")
```



Lab exercise 4/4 (Laengelmavesi revisited)

Use the data in `Laengelmavesi2.csv` (only) to create the following graphs. Make sure to add axis labels and plot titles.

1. Plot the mean weight of each species as a function of the mean length, with the species names and mean heights also indicated on the plot.
2. Create a boxplot and histograms of the length distributions for each species. The whiskers on the boxplot should be solid lines.
3. Plot all the weights vs the lengths. Include enough information that the data for each species can be identified. Indicate the mean weight, height, and length for each species.
4. Create one plot of the heights as a function of the lengths. Add a line separating fish with height greater than 20cm. Place the tick marks and tick mark labels inside the plot box.

Next time...

1/27: Introduction to R and R Studio, working with data
2/03: Lists, Intro to plotting, manipulating data
2/10: Probability, linear modeling, PCA
2/17: Programming practices, conditional statements
2/24: Creating functions, debugging
3/02: Permutation analysis
3/09: Advanced plotting

Saving plots to file

Eventually you will want to do something with the plots you create.

Journals almost always prefer Figures as separate files.

RStudio has ways of exporting figures, but because of the GUI interface these sometimes don't end up looking the way you intended them.

We can tell `R` to send plots to a file rather than plotting in the GUI.

First step, decide on a format:

Format	Driver	Notes
JPG	`jpeg`	Can be used anywhere, but doesn't resize
PNG	`png`	Can be used anywhere, but doesn't resize
WMF	`win.metafile`	Windows only; best choice with Word; easily resizable
PDF	`pdf`	Best choice with pdflatex; easily resizable
Postscript	`postscript`	Best choice with latex and Open Office; easily resizable

General Method

1. Decide on a format.
2. Before the `R` code that plots your graph, insert a call to the relevant driver.
3. After plotting, enter `dev.off()`

This plots the figure to the file you specified.
Note that you won't see the figure in the Rstudio window.

e.g. output to pdf

```
>   pdf(file="output.pdf", width=8, height=4)
>   #code to plot your world-changing results
>   dev.off()
pdf
  2
```

Calls to drivers

The different graphics devices have their own sets of options. You can customize figure size/resolution etc.

```
> pdf(file = ifelse(onefile, "Rplots.pdf",
+     "Rplot%03d.pdf"), width, height, onefile,
+     family, title, fonts, version,paper,
+     encoding, bg, fg, pointsize, pagecentre,
+     colormodel,useDingbats, useKerning,
+     fillOddEven, compress)
>
> png(filename = "Rplot%03d.png",width = 480,
+     height = 480, units = "px", pointsize = 12,
+     bg = "white",  res = NA, ...,
+     type = c("cairo", "cairo-png", "Xlib",
+              "quartz"), antialias)
```

More on graphics devices

I generally use pdf, you can plot multiple graphs to a single file. (file won't be created until you call `dev.off()`)

.png is useful, and some journals require tiff or postscript files.

Sometimes window size or cex changes can result in text and points being of different relative sizes when output using different graphics devices.

Stick with one format and optimize that for display (e.g. pdf).

Won't solve all your problems, but that's what cats (or bacon) are for.