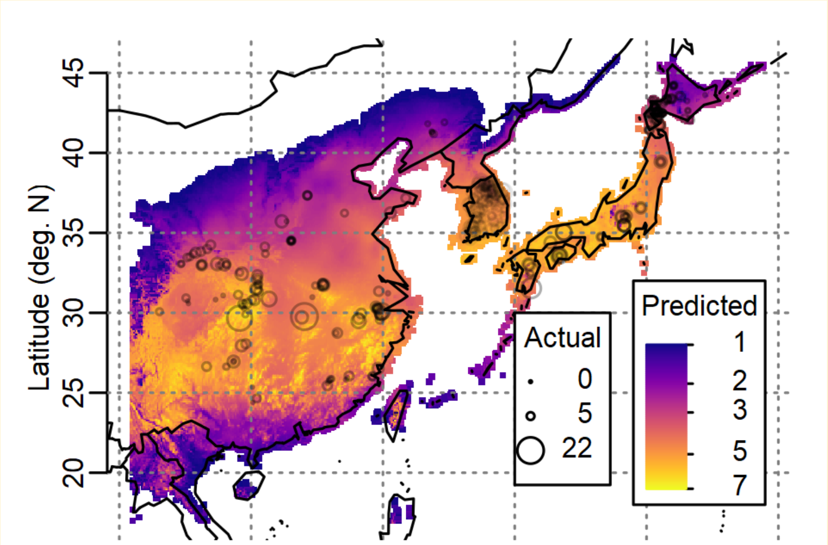
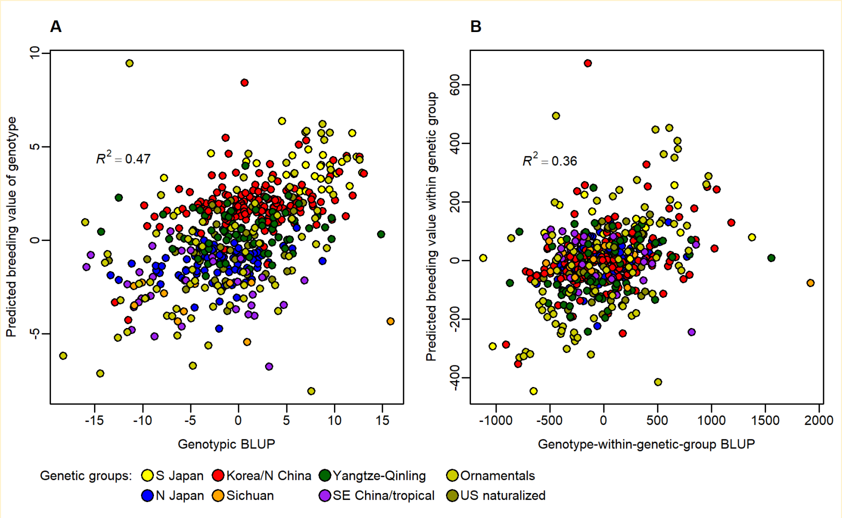
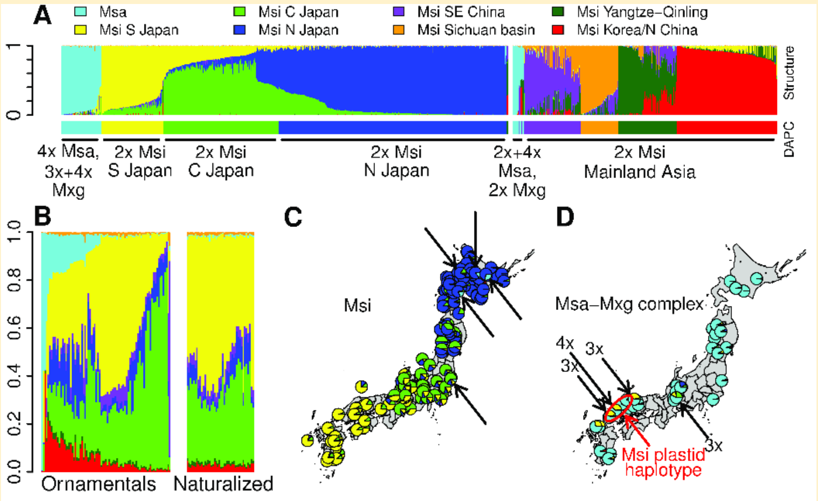




Graphics in R

CPSC 499, Lecture 3, Fall 2018

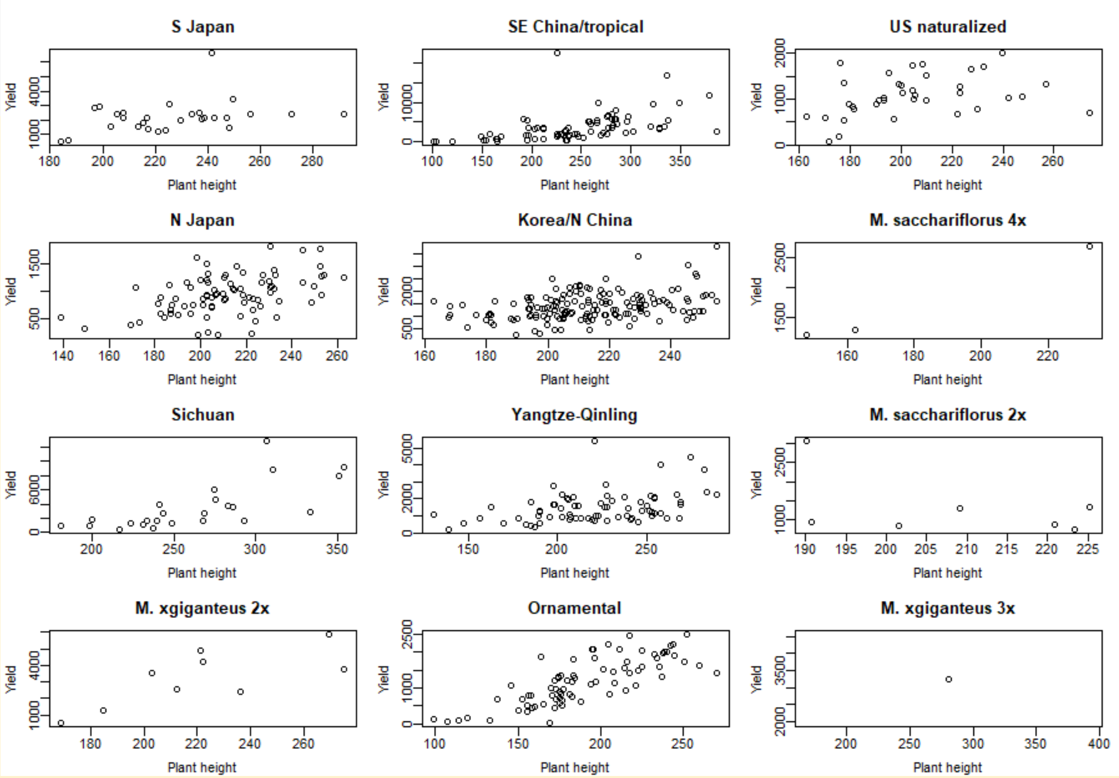


Packages for making graphics

- `graphics`
  - Installed with R
  - Very versatile
  - Not very user-friendly; tasks like “color points based on a grouping factor” must be done manually
- `lattice`
  - Useful for making multiple plots based on a grouping factor
- `ggplot2`
  - Integrated with Tidyverse
  - Very user-friendly exploration of datasets
  - Not as good when you need to add something custom to a plot

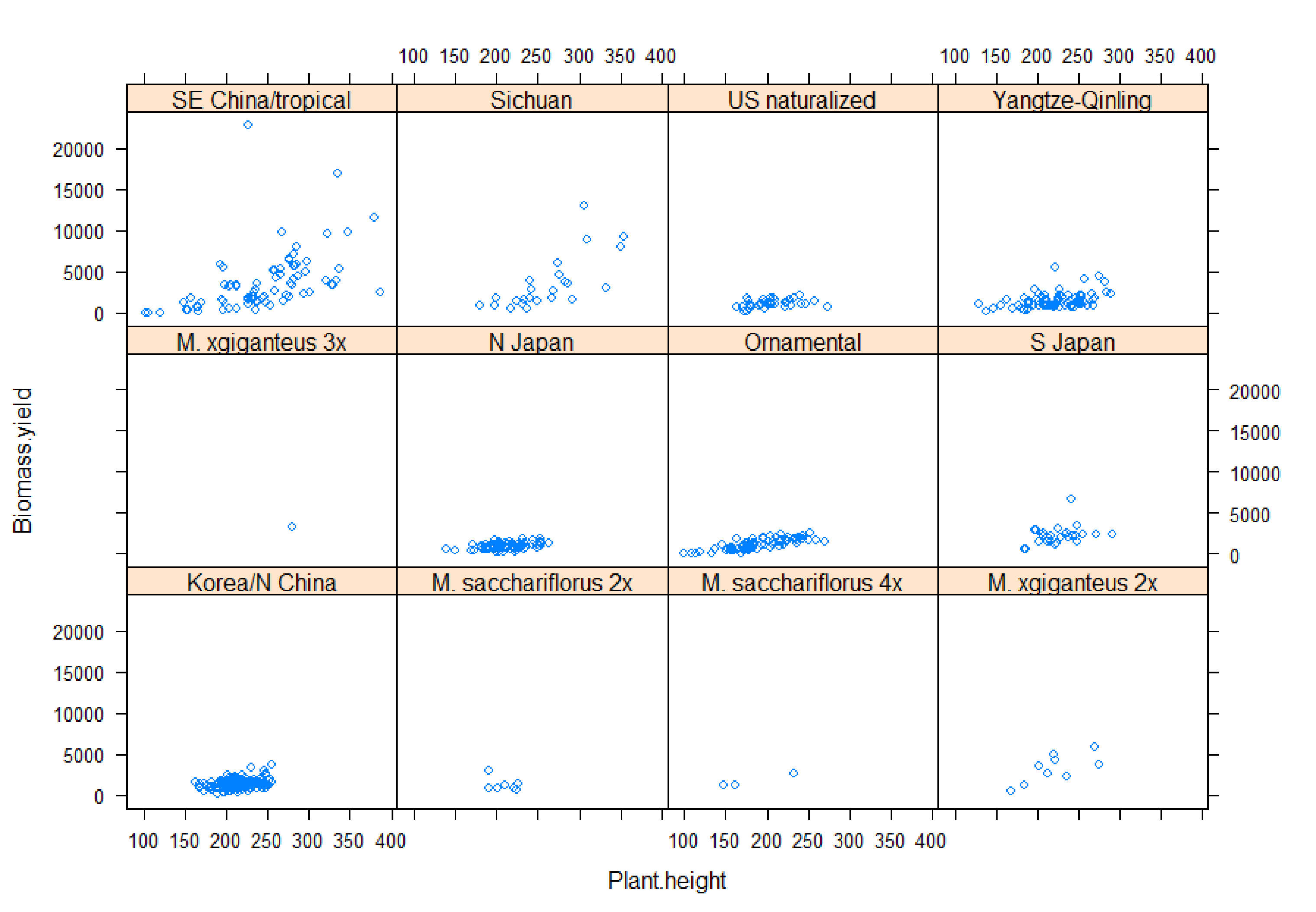
Yield vs. height by group with `graphics`

```
par(mfrow = c(4, 3), mar = c(3.1, 3.1, 3.1, 1.1),
    mgp = c(2, 0.5, 0))
mygrp <- unique(mydata$Genetic.group)
for(g in mygrp){
  gsubset <- which(mydata$Genetic.group == g)
  plot(mydata$Plant.height[gsubset],
       mydata$Biomass.yield[gsubset],
       xlab = "Plant height", ylab = "Yield",
       main = g)
}
```



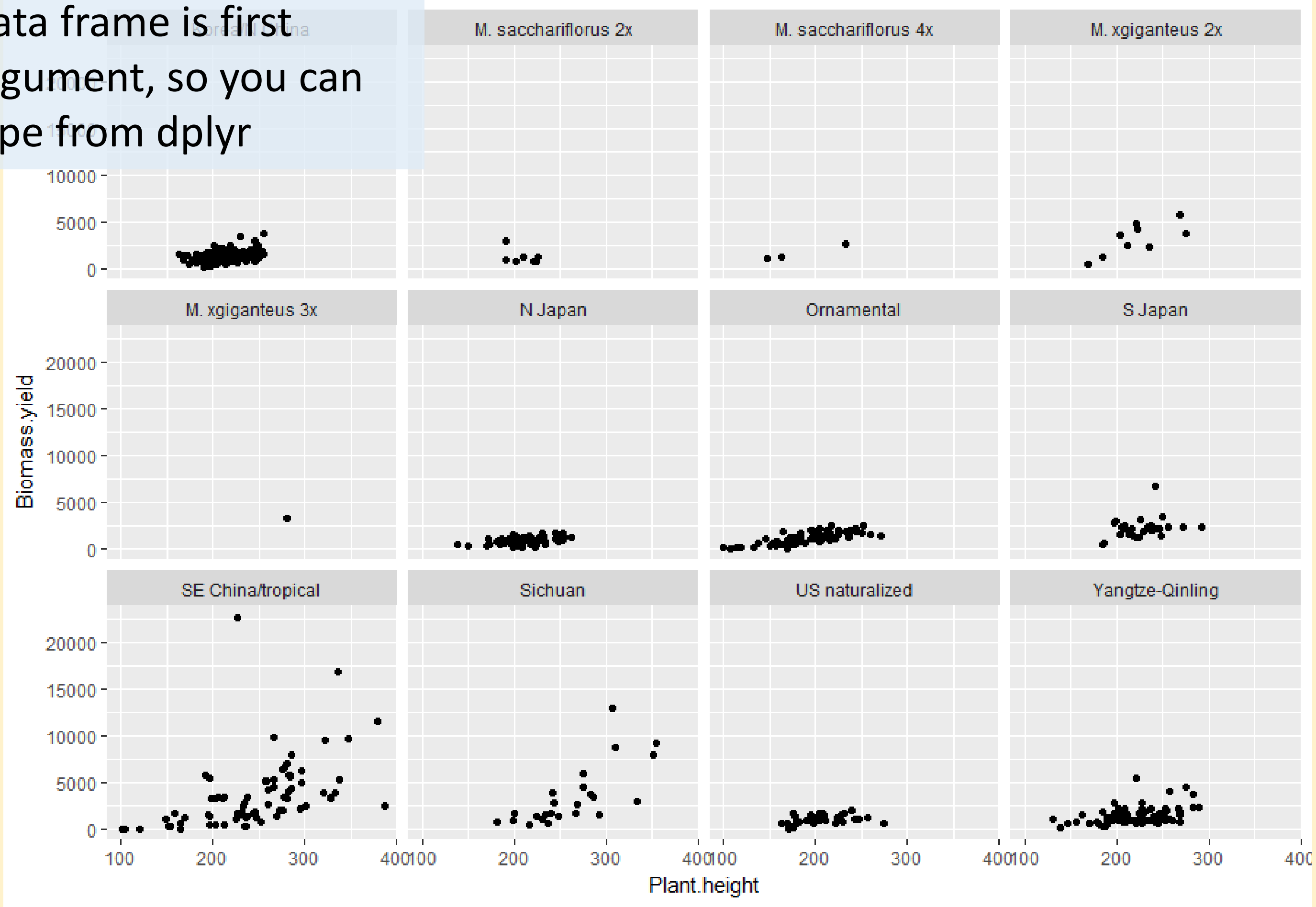
Yield vs. height by group with `lattice`

```
xyplot(Biomass.yield ~ Plant.height | Genetic.group, mydata)
```



Yield vs. height by group with `ggplot2`

```
ggplot(mydata, aes(x = Plant.height, y = Biomass.yield)) +
  geom_point() +
  facet_wrap(~ Genetic.group)
```



Data frame is first argument, so you can pipe from dplyr

Syntax of `ggplot2`: the `aes` function

- Something specific to `ggplot2` called *aesthetic mappings*
- Shows which aspects of the plot correspond to which variables
- Also can set things to a single value, e.g. make all points red, if you set them outside the `aes` function
- Some common arguments:
  - x, y
  - color
  - shape
  - size

Syntax of `ggplot2`: `ggplot` function

- First argument is a data frame with the data you want to plot
- Second argument (optional) is a call to `aes` indicating mappings that will be universal across all layers
- This function is mandatory, but by itself will not create a plot
- One of several function calls that should be connected with + signs

Syntax of `ggplot2`: `geom` functions

- These tell ggplot what kind of plot to make
- You can call several of them to put layers on top of each other, like a scatter plot and a trend line
- First argument is a call to `aes` indicating any mappings specific to that layer
- All start with `geom_`
- Type `?geom` to get a list

Syntax of `ggplot2`: `facet` functions

- For making multiple plots based on one or more grouping variables
- `facet_wrap`: If you have one grouping variable
  - Pass it a formula like `~ var1`
- `facet_grid`: If you have two grouping variables
  - Pass it a formula like `var2 ~ var1`

Syntax of `ggplot2`: `coord` functions

- For changing the coordinate system of the plot
- `coord_flip`: move x to y and y to x (useful for box plots and violin plots)
- `coord_trans`: e.g. if you need to log-transform an axis
- Others for more obscure situations (pie charts, world maps)

Mini-exercise

- Create a new call to `ggplot(mydata)`
- Try out `geom_boxplot`, with `x = Genetic.group` and `y = Number.of.stems`
- Use `coord_flip` to rotate the plot
- Look at the Aesthetics section of `?geom_boxplot`
- Try `fill = Genetic.group`

Syntax of `ggplot2`: `scale` functions

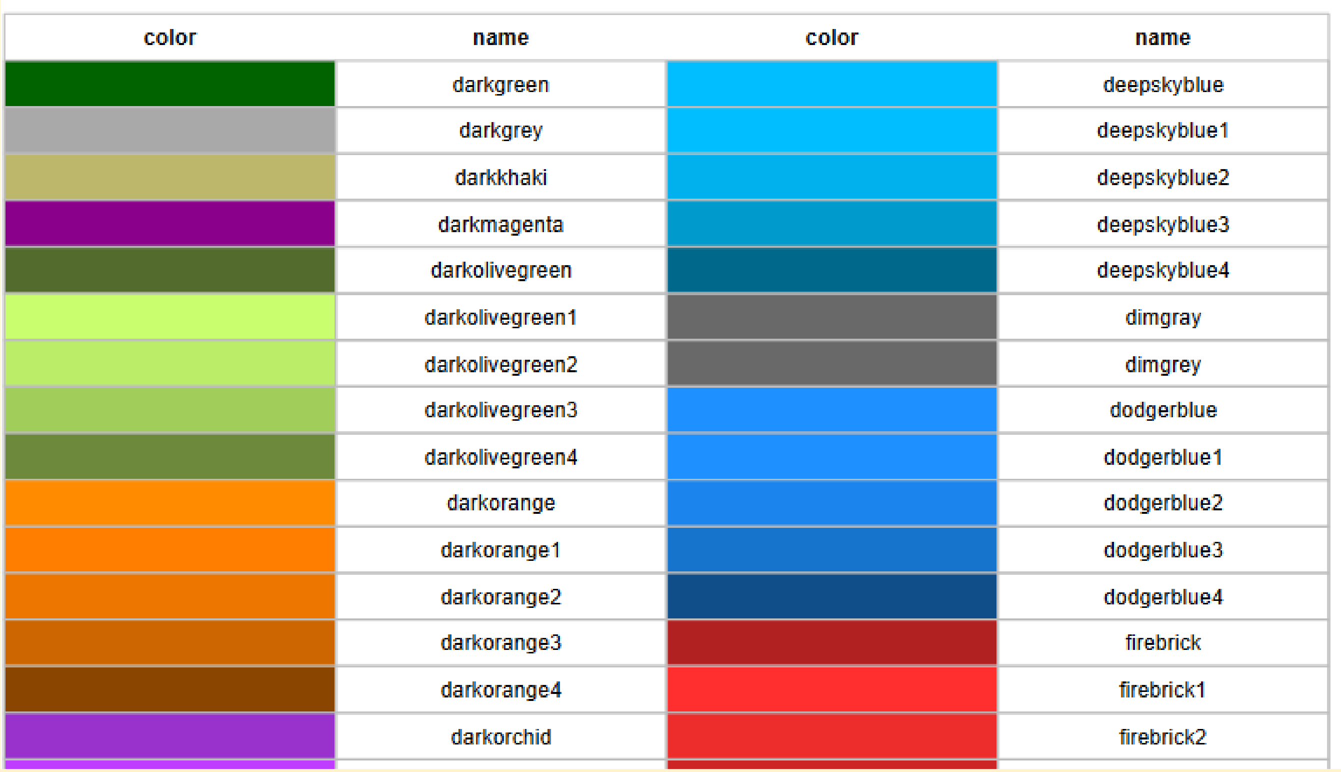
- For when you want more control over aesthetic mappings
- E.g. you need each genetic group to be a specific color or shape
- Or you just want to use a specific, pre-made scheme (more on these later)

The R base graphics system

(with some aspects that also apply to ggplot2)

Colors in R

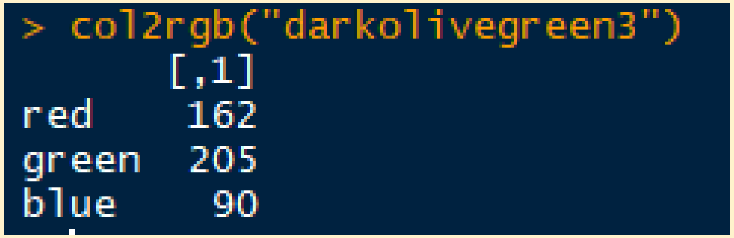
- Many are named and can be specified using the corresponding character string.
- Arguments `col`, `fg`, `bg`, `fill`, accept these
- See http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf



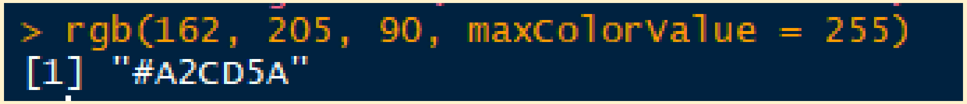
color	name	color	name
🟩	darkgreen	🟦	deepskyblue
◼	darkgrey	🟦	deepskyblue1
🟨	darkkhaki	🟦	deepskyblue2
🟪	darkmagenta	🟦	deepskyblue3
🟩	darkolivegreen	🟦	deepskyblue4
🟩	darkolivegreen1	◼	dimgray
🟩	darkolivegreen2	◼	dimgrey
🟩	darkolivegreen3	🟦	dodgerblue
🟩	darkolivegreen4	🟦	dodgerblue1
🟧	darkorange	🟦	dodgerblue2
🟧	darkorange1	🟦	dodgerblue3
🟧	darkorange2	🟦	dodgerblue4
🟧	darkorange3	🟥	firebrick
🟫	darkorange4	🟥	firebrick1
🟪	darkorchid	🟥	firebrick2

Colors in R

- If you know what RGB values you want (for example to match colors from other software) use the `rgb` function.
- Generates a string like: "#14C896" (hexadecimal)
- You can use the `alpha` argument to set transparency
- The `col2rgb` function can take a named color and give you the RGB values for it



```
> col2rgb("darkolivegreen3")
      [,1]
red    162
green  205
blue    90
```



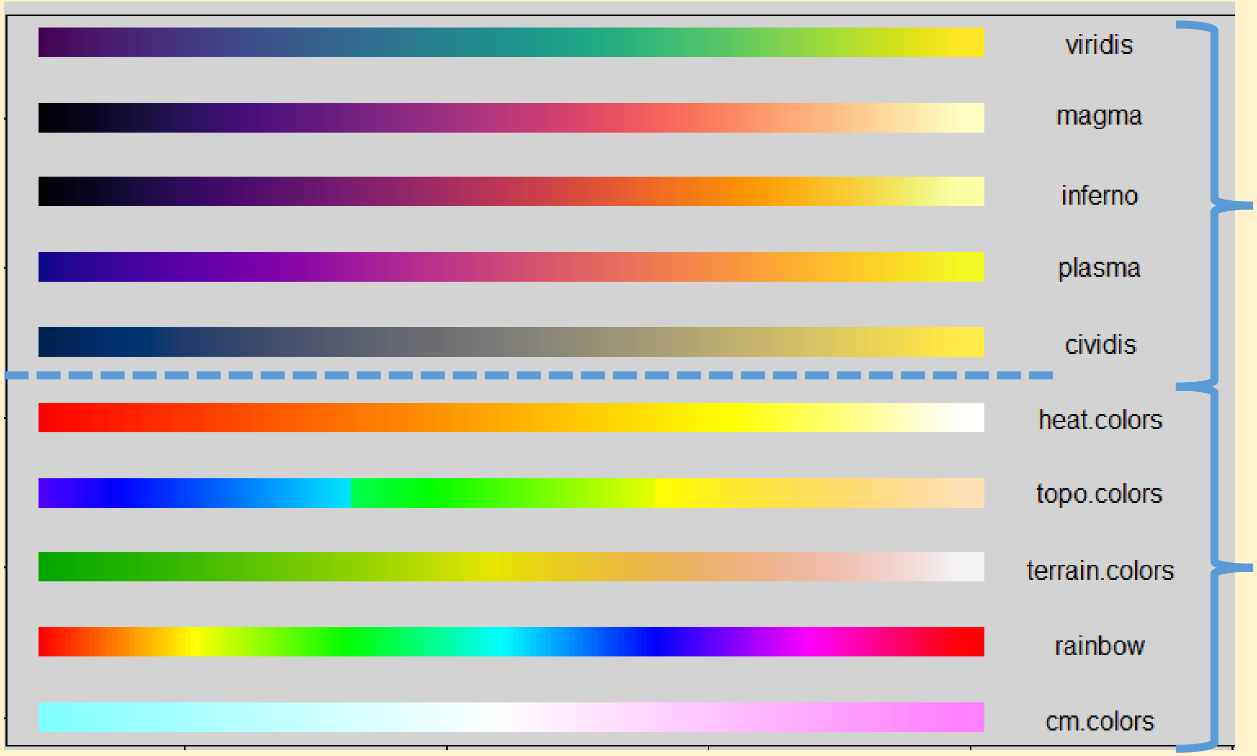
```
> rgb(162, 205, 90, maxColorValue = 255)
[1] "#A2CD5A"
```

Mini-exercise

- On a scale of 0-255:
  - Illini Blue = Red 19, Green 41, Blue 75
  - Illini Orange = Red 232, Green 74, Blue 39
- Use `rgb` to make strings representing these two colors
- Pass them to the `col` argument in a scatter plot to color points using them

Color palettes

- If you need to generate a series of colors
- E.g. `rainbow(100)` makes a vector of 100 colors
- See `scale_color_viridis` for ggplot2

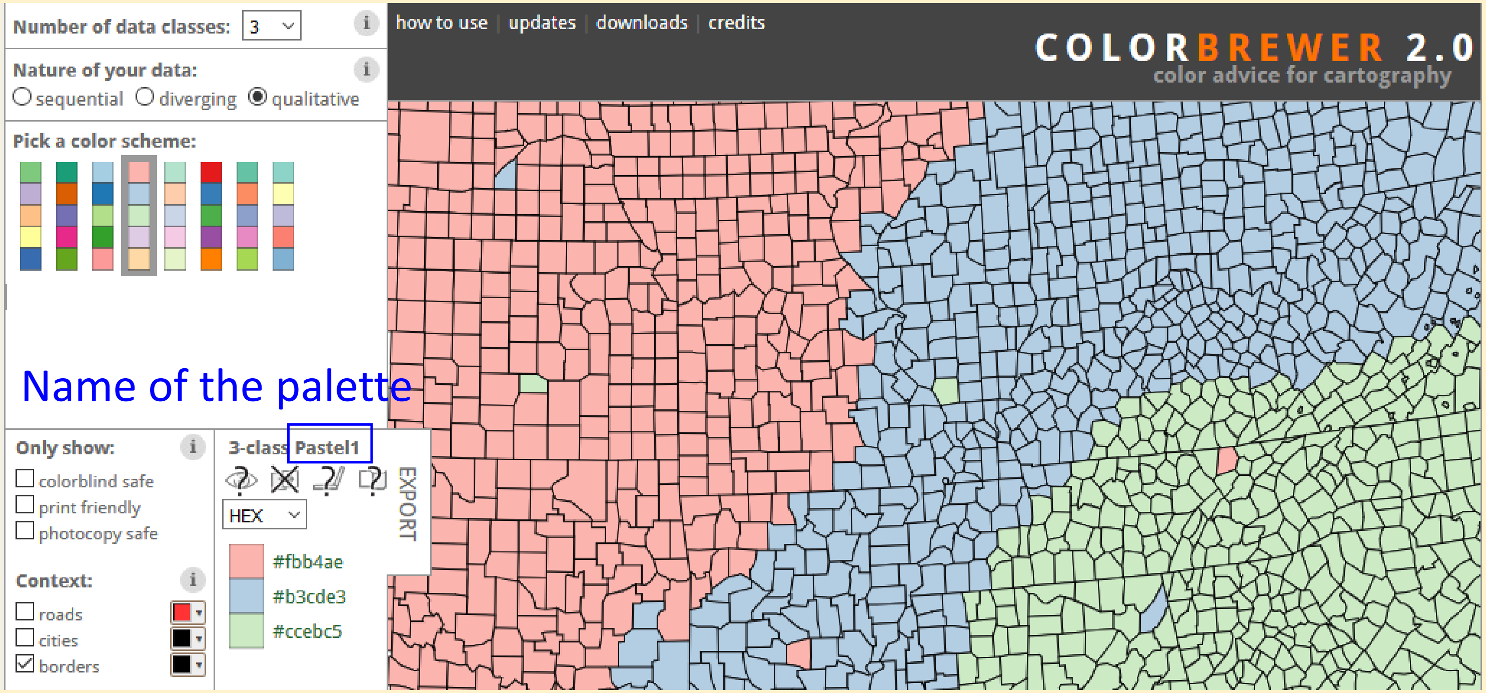


Available in `viridis` package.
R-G color blind friendly.
Print well in greyscale.

Installed with R

Color Brewer: qualitative sets of colors

- http://colorbrewer2.org/
- Good for categorical data
- Available in `RColorBrewer` package
- See `scale_color_brewer` for ggplot2



Number of data classes: 3

Nature of your data:
○ sequential ○ diverging ● qualitative

Pick a color scheme:

Name of the palette

Only show:
□ colorblind safe
□ print friendly
□ photocopy safe

Context:
□ roads
□ cities
☑ borders

3-class Pastel1

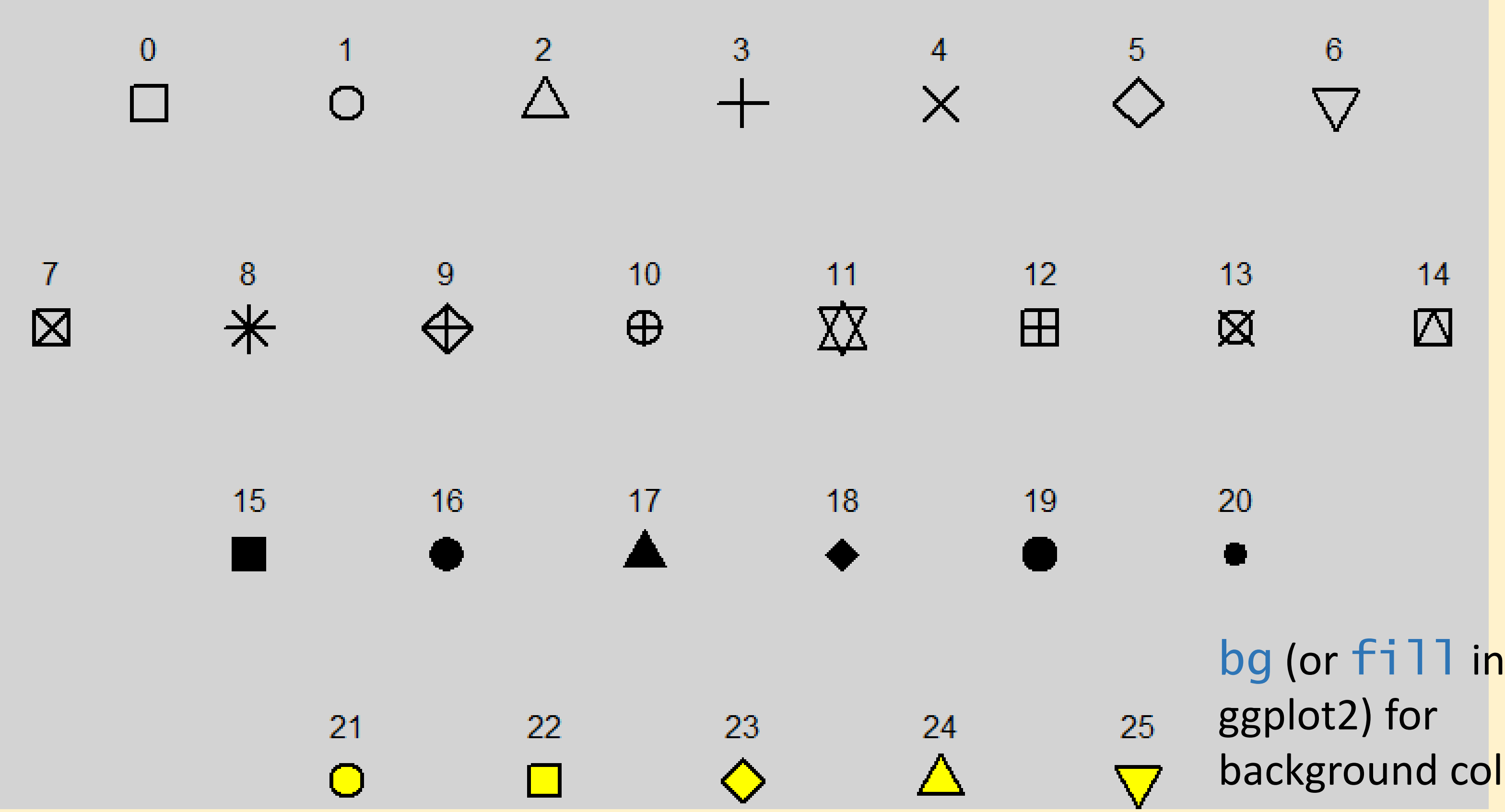
HEX

#fbb4ae
#b3cde3
#ccebc5

how to use | updates | downloads | credits
COLORBREWER 2.0
color advice for cartography
EXPORT

Point shapes in R

- pch argument in the base plotting system
- shape argument in ggplot2



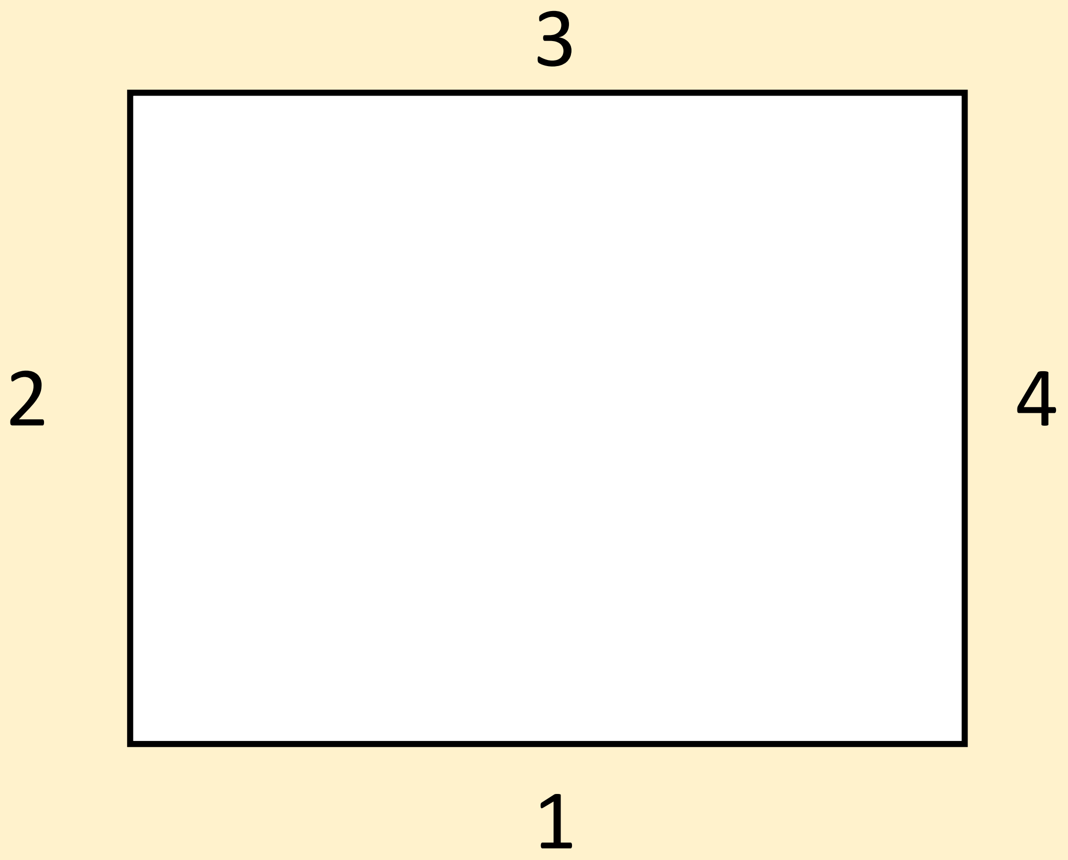
bg (or `fill` in ggplot2) for background color

Exploring other graphical parameters: the `par` function

- See `?par` for a very long list
- Call `par` to set parameters universally, e.g. `par(lwd = 2)` (set line width to 2)
- Many can also be used as arguments to `plot`, `points`, `text`, and other plotting functions
- `par("lwd")` would give current value

`mar`	Plot margin	`las`	Axis label rotation
`mgp`	Label distance from axis	`lty`	Dotted or dashed lines
`cex`	Character magnification	`xlog`	Log transform x-axis
`family`	Font name	`mfrow` `mfcol`	Put multiple plots in one graphics device
`font`	Bold, italic		

Plot sides are identified by number



- Set margin sizes with **`par(mar)`** in this order

- Set **`axes = FALSE`** and then use the **`axis`** function to manually add custom axes; first argument is this number

Adding legends with the `legend` function

- Unlike with ggplot2, you build legend manually and decide what goes in it
- First two arguments are xy coords on plot
- Third argument is vector of names of categories that should show up
- Points, lines, or boxes will show up next to names
- Arguments specify things like color, shape, fill color, line type

Adding a trendline: `abline`

- Can specify `a` and `b` for intercept and slope
- Or give values to `v` or `h` for vertical or horizontal lines
- Or, make a linear regression model with `lm` and pass that to `abline`

```
plot(var1, var2)
abline(lm(var2 ~ var1))
```

Drawing on a plot

- `rect` to draw a rectangle
- `segments` to draw line segments
- `arrows` to draw arrows
- `points` to add points to a plot
- `text` to add text to particular xy coordinates
- `locator` to click on a plot and get the coordinates

Mini exercise

- Use the arrow and locator functions to create a new function:
- After the function is called, it lets you click on two points and then draws an arrow between them
- (Hint: the function does not need to have any arguments)

More handy functions: `plotrix` package

- `draw.circle` and `draw.ellipse`
- `floating.pie` – put a pie chart at any x-y coordinate
- many others

Formatting text: `expression`

- Italics, subscripts, etc.
- Mathematical symbols
- `?plotmath` to see your options
- Intended for writing mathematical expressions; can be hacked for other purposes but you may have to experiment

Adding other special characters

- Unicode – put “\u” plus four digits in a text string
- E.g. “jalape\u00F1o” = jalapeño
- (Google “Unicode tables” to look up others)

- `emojifont` package on CRAN

- Use `windowsFonts` function to get access to your system fonts on Windows

Plotting directly to a file

- Generally want to test out in RStudio first
- Functions to open file connection:
  - `pdf`, `postscript`
  - `cairo_pdf`, `cairo_ps`, `svg` – better for unusual fonts and characters
  - `tiff`, `bmp`, `jpeg`, `png`
  - arguments for size, resolution, font
- Then do all commands to create your plot
- `dev.off()` to close the file connection (finish the plot)

Mini exercise

- Use `pdf` to open a file for writing plots
- Run the plot command several times to make several plots
- Use `dev.off` to finish the file
- What does the file look like?