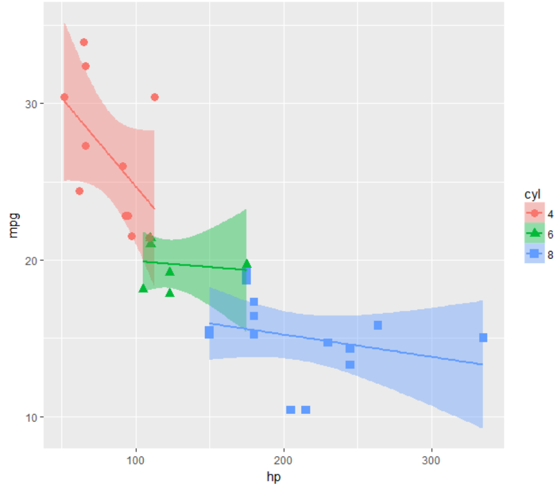
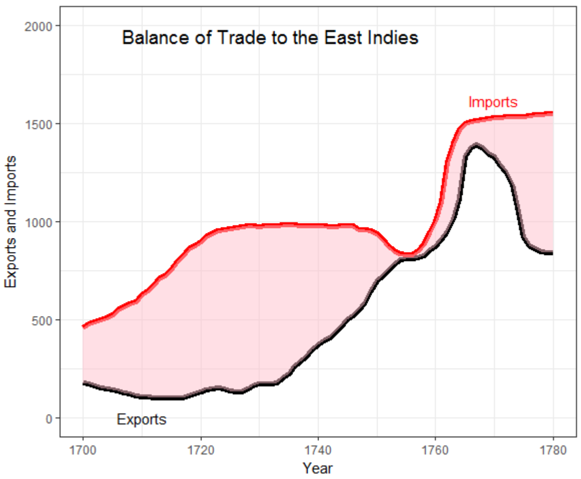
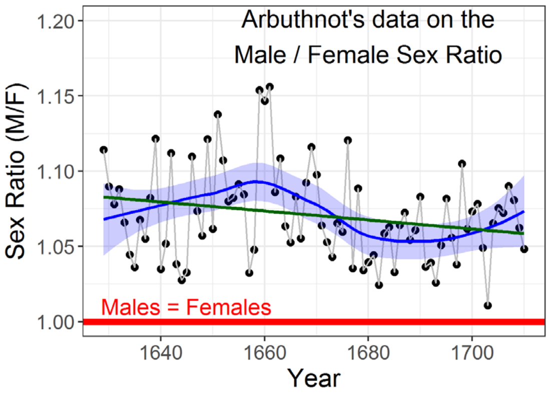






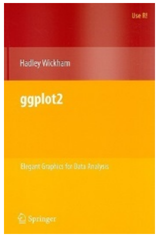
Introduction to ggplot2

Michael Friendly

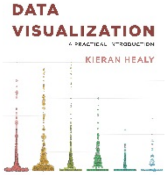
Psych 6135

http://euclid.psych.yorku.ca/www/psy6135/

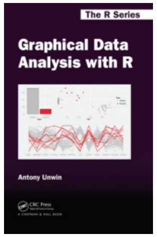
Resources: Books



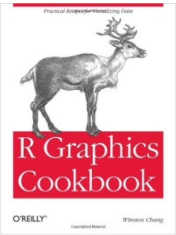
Hadley Wickham, *ggplot2: Elegant graphics for data analysis*, 2nd Ed.
1st Ed: Online, http://ggplot2.org/book/
ggplot2 Quick Reference: http://sape.inf.usi.ch/quick-reference/ggplot2/
Complete ggplot2 documentation: http://docs.ggplot2.org/current/



Kieran Healy, *Data Visualization, a Practical Introduction*
A hands-on introduction to data visualization using ggplot2, with a wide range of topics.
The online version: https://socviz.co/ is a great example of R bookdown publishing.



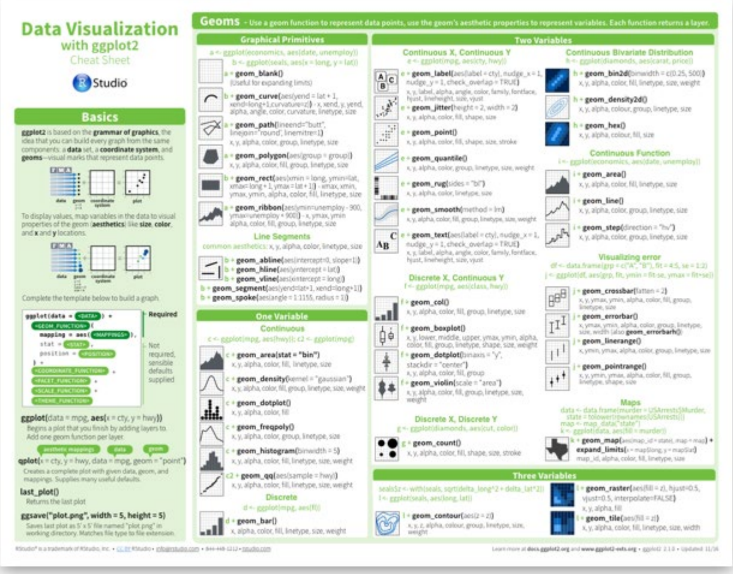
Antony Unwin, *Graphical Data Analysis with R*
A gentile introduction to doing visual data analysis, mainly with ggplot2.
R code: http://www.gradaanwr.net/



Winston Chang, *R Graphics Cookbook: Practical Recipes for Visualizing Data*
Cookbook format, covering common graphing tasks; the main focus is on ggplot2
R code from book: http://www.cookbook-r.com/Graphs/
Download from: http://ase.tufts.edu/bugs/guide/assets/R%20Graphics%20Cookbook.pdf

Resources: Cheat sheets

- R Studio maintains a large number of cheat sheets, https://www.rstudio.com/resources/cheatsheets/
- Topics:
  - R Studio IDE, Data import, Data transformation (dplyr), Data visualization (ggplot2), R Markdown, …
  - My collection: R Studio Cheat Sheets



Data Visualization with ggplot2 Cheat Sheet

RStudio

Geoms - Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

Basics

ggplot2 is based on the grammar of graphics, the idea that you can build every graph from the same components: a data set, a coordinate system, and geoms—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (aesthetics) like size, color, and x and y locations.

Complete the template below to build a graph.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
    mapping = aes(<MAPPINGS>),
    stat = <STAT>,
    position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

Required

Not required, sensible defaults supplied

ggplot(data = mpg, aes(x = cty, y = hwy))
Begins a plot that you finish by adding layers to. Add one geom function per layer.

qplot(x = cty, y = hwy, data = mpg, geom = "point")
Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot()
Returns the last plot

ggsave("plot.png", width = 5, height = 5)
Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Graphical Primitives

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z)) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group))
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin alpha, color, fill, group, linetype, size

Line Segments

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))
b + geom_segment(aes(yend=lat+1, xend=long+1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

One Variable

Continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly()
x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5)
x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy))
x, y, alpha, color, fill, linetype, size, weight

Discrete

d <- ggplot(mpg, aes(fl))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

Two Variables

Continuous X, Continuous Y

e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE)
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

e + geom_point()
x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile()
x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl")
x, y, alpha, color, linetype, size

e + geom_smooth(method = lm)
x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE)
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

f <- ggplot(mpg, aes(class, hwy))

f + geom_col()
x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot()
x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill, group

f + geom_violin(scale = "area")
x, y, alpha, color, fill, group, linetype, size, weight

Discrete X, Discrete Y

g <- ggplot(diamonds, aes(cut, color))

g + geom_count()
x, y, alpha, color, fill, shape, size, stroke

Continuous Bivariate Distribution

h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

Continuous Function

i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar()
x, ymax, ymin, alpha, color, group, linetype, size, width (also geom_errorbarh())

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map) + expand_limits(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
l <- ggplot(seals, aes(long, lat))

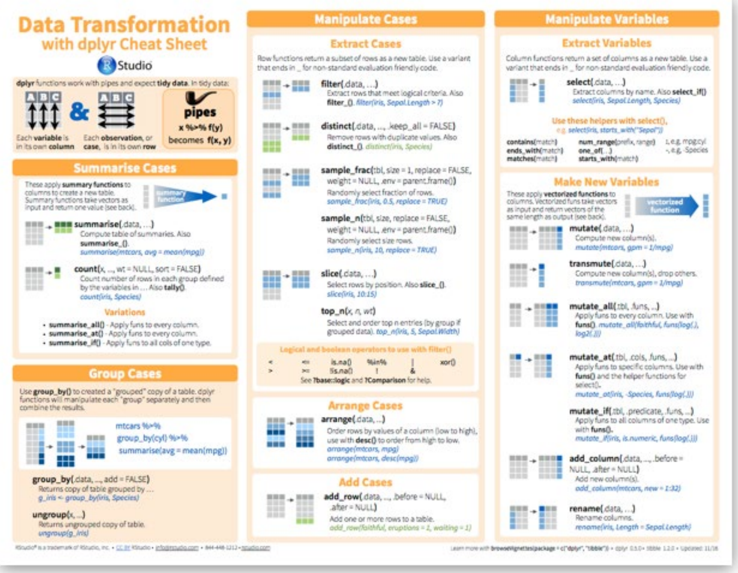
l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z))
x, y, alpha, color, fill, linetype, size, width

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

Learn more at docs.ggplot2.org and www.ggplot2-exts.org • ggplot2 2.1.0 • Updated: 11/16



Data Transformation with dplyr Cheat Sheet

RStudio

dplyr functions work with pipes and expect tidy data. In tidy data:

Each variable is in its own column & Each observation, or case, is in its own row

pipes
x %>% f(y) becomes f(x, y)

Summarise Cases

These apply summary functions to columns to create a new table. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries. Also summarise_().
summarise(mtcars, avg = mean(mpg))

count(x, …, wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also tally().
count(iris, Species)

Variations

- summarise_all() - Apply funs to every column.
- summarise_at() - Apply funs to every column.
- summarise_if() - Apply funs to all cols of one type.

Group Cases

Use group_by() to created a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

group_by(.data, …, add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

Extract Cases

Row functions return a subset of rows as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

filter(.data, …)
Extract rows that meet logical criteria. Also filter_(). filter(iris, Sepal.Length > 7)

distinct(.data, …, .keep_all = FALSE)
Remove rows with duplicate values. Also distinct_(). distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame())
Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame())
Randomly select size rows.
sample_n(iris, 10, replace = TRUE)

slice(.data, …)
Select rows by position. Also slice_().
slice(iris, 10:15)

top_n(x, n, wt)
Select and order top n entries (by group if grouped data). top_n(iris, 5, Sepal.Width)

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See ?base::logic and ?Comparison for help.

Arrange Cases

arrange(.data, …)
Order rows by values of a column (low to high), use with desc() to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

Add Cases

add_row(.data, …, .before = NULL, .after = NULL)
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

Extract Variables

Column functions return a set of columns as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

select(.data, …)
Extract columns by name. Also select_if()
select(iris, Sepal.Length, Species)

Use these helpers with select(),
e.g. select(iris, starts_with("Sepal"))

contains(match)	num_range(prefix, range)	:, e.g. mpg:cyl
ends_with(match)	one_of(…)	-, e.g. -Species
matches(match)	starts_with(match)	

Make New Variables

These apply vectorized functions to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …)
Apply funs to every column. Use with funs(). mutate_all(faithful, funs(log(.), log2(.)))

mutate_at(.tbl, .cols, .funs, …)
Apply funs to specific columns. Use with funs() and the helper functions for select().
mutate_at(iris, -Species, funs(log(.)))

mutate_if(.tbl, .predicate, .funs, …)
Apply funs to all columns of one type. Use with funs().
mutate_if(iris, is.numeric, funs(log(.)))

add_column(.data, …, .before = NULL, .after = NULL)
Add new column(s).
add_column(mtcars, new = 1:32)

rename(.data, …)
Rename columns.
rename(iris, Length = Sepal.Length)

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.5.0 • tibble 1.2.0 • Updated: 11/16

What is ggplot2?

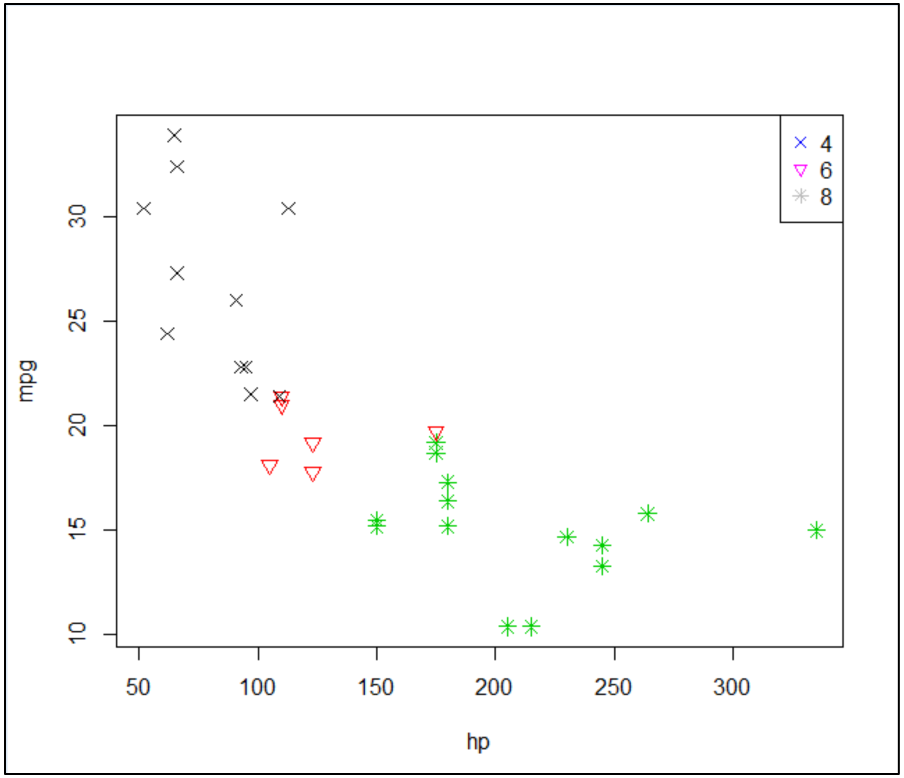
- ggplot2 is Hadley Wickham's R package for producing "elegant graphics for data analysis"
  - An implementation of the ideas for graphics introduced in Lee Wilkinson's *Grammar of Graphics*
  - These ideas and the syntax of ggplot2 help to think of graphs in a new and more general way
  - Produces pleasing plots, taking care of many of the fiddly details (legends, axes, colors, …)
  - It is built upon the "grid" graphics system
  - It is open software, with a large number of gg_ extensions. See: https://exts.ggplot2.tidyverse.org/gallery/

ggplot2 vs base graphics

Some things that should be simple are harder than you'd like in base graphics

Here, I'm plotting gas mileage (mpg) vs. horsepower and want to use color and shape for different # of cylinders.

But I don't quite get it right!



```
mtcars$cyl <- as.factor(mtcars$cyl)
plot(mpg ~ hp , data=mtcars,
     col=cyl, pch=c(4,6,8)[mtcars$cyl], cex=1.2)
legend("topright", legend=levels(mtcars$cyl),
          pch = c(4,6,8),
          col=levels(mtcars$cyl))
```

colors and point symbols work differently in plot() and legend()

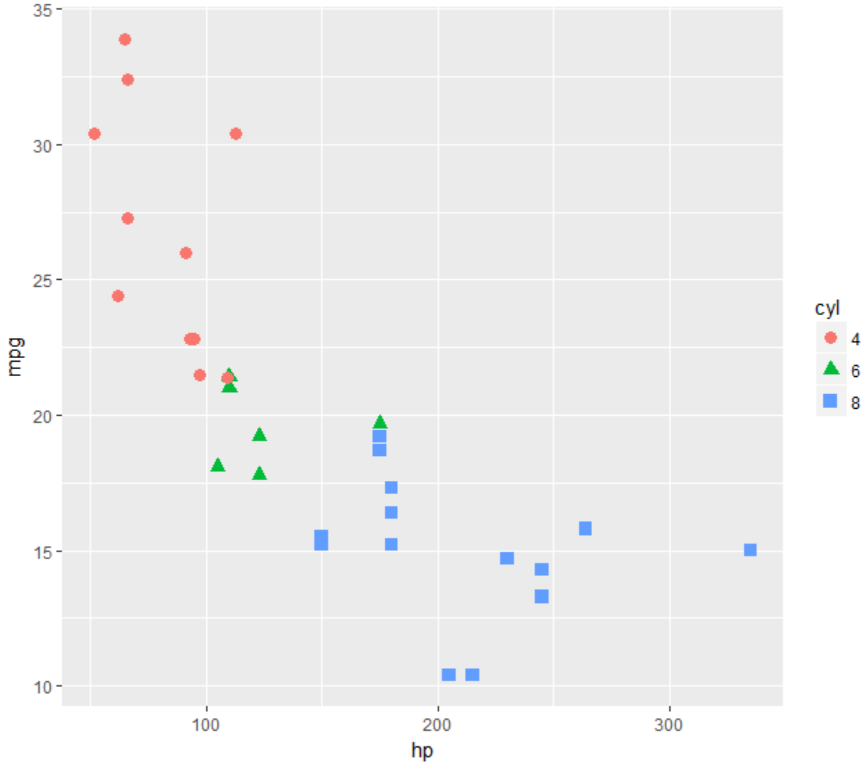
goal of ggplot2: this should "just work"

ggplot2 vs base graphics

In ggplot2, just map the data variables to aesthetic attributes
  aes(x, y, shape, color, size, …)

ggplot() takes care of the rest

aes() mappings set in the call to ggplot() are passed to geom_point() here

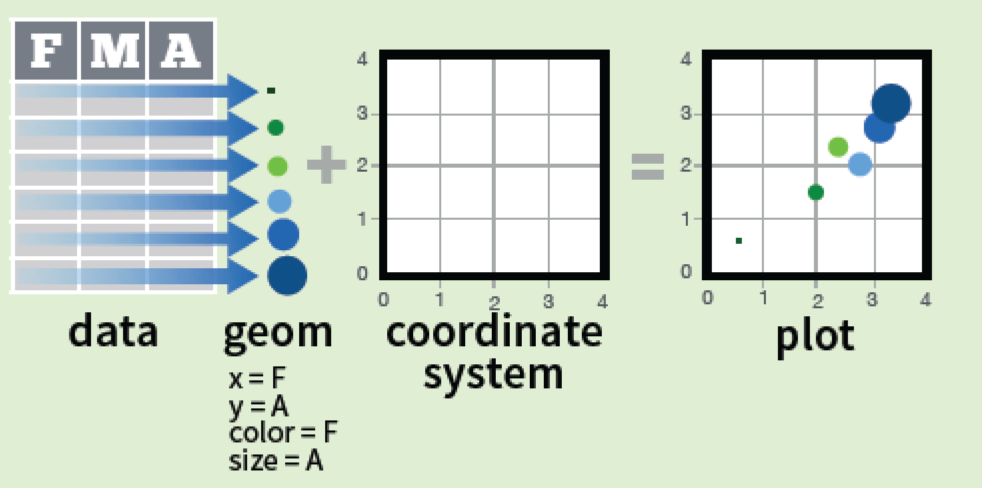


```
library(ggplot2)
ggplot(mtcars, aes(x=hp, y=mpg, color=cyl, shape=cyl)) +
    geom_point(size=3)
```

Follow along: the R script for this example is at: http://euclid.psych.yorku.ca/www/psy6135/R/gg-cars.R

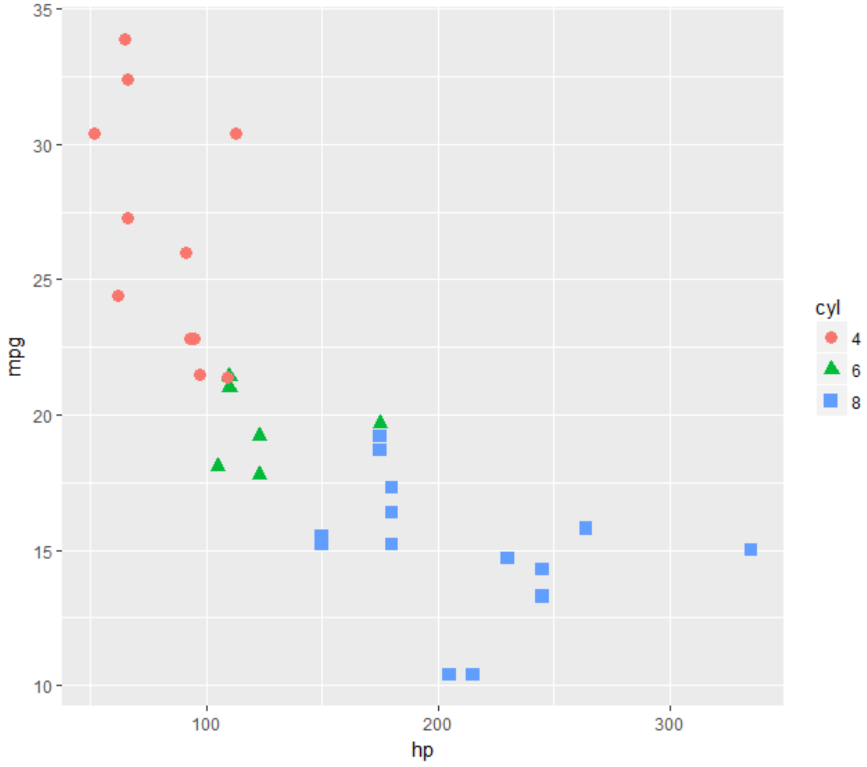
Grammar of Graphics

- Every graph can be described as a combination of independent building blocks:
  - data: a data frame: quantitative, categorical; local or data base query
  - aesthetic mapping of variables into visual properties: size, color, x, y
  - geometric objects (“geom”): points, lines, areas, arrows, …
  - coordinate system (“coord”): Cartesian, log, polar, map,



ggplot2: data + geom -> graph

```
ggplot(data=mtcars,                          ❶
          aes(x=hp, y=mpg,                   ❷
                color=cyl, shape=cyl)) +     ❸
    geom_point(size=3)                       ❹
```



In this call,

1. data=mtcars: data frame
2. aes(x=hp, y=mpg): plot variables
3. aes(color, shape): attributes
4. geom_point(): what to plot

- the coordinate system is taken to be the standard Cartesian (x,y)

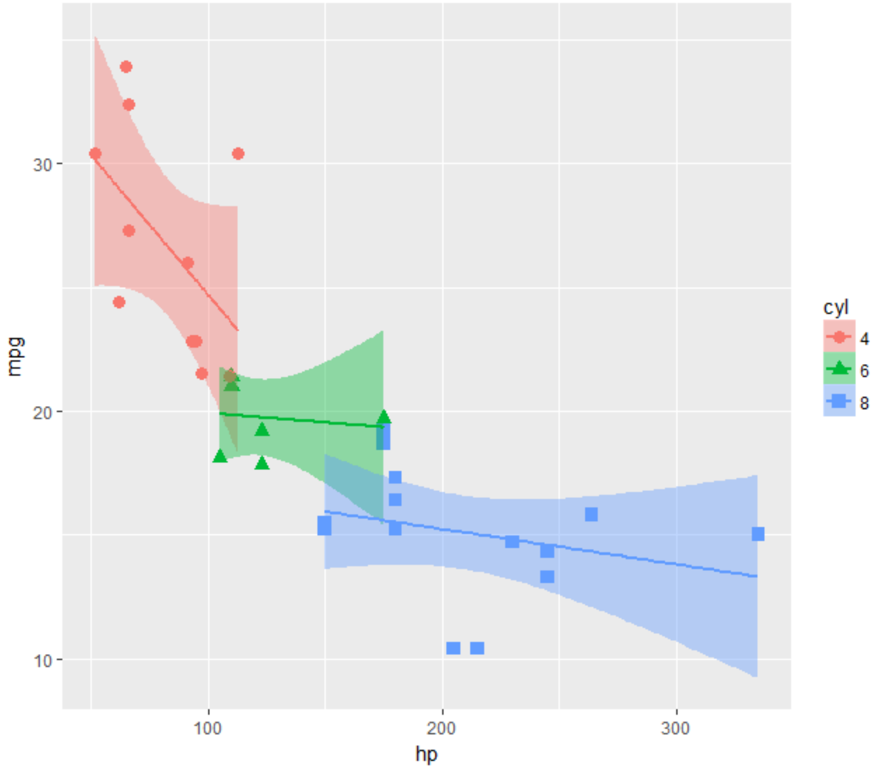
ggplot2: geoms

Wow! I can really see something there.

How can I enhance this visualization?

Easy: add a geom_smooth() to fit linear regressions for each level of cyl

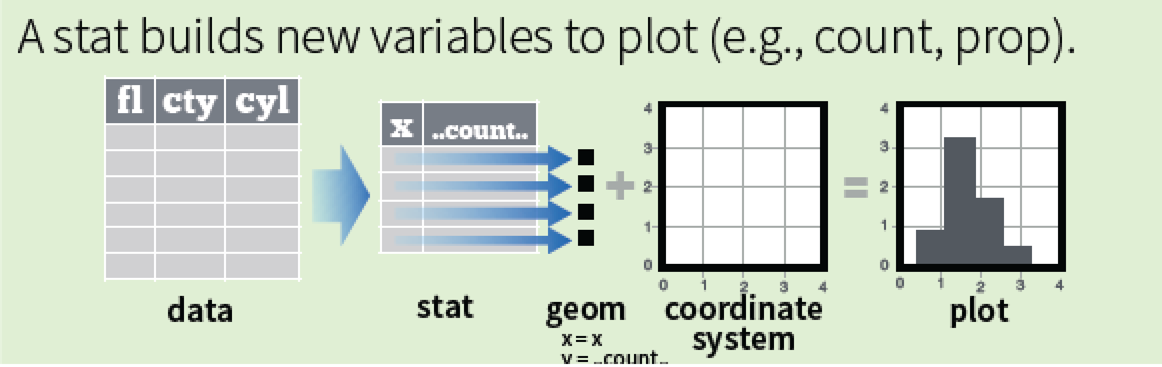
It is clear that horsepower and # of cylinders are highly related (Duh!)



```
ggplot(mtcars, aes(x=hp, y=mpg, color=cyl, shape=cyl)) +
   geom_point(size=3) +
   geom_smooth(method="lm", aes(fill=cyl))
```

Grammar of Graphics

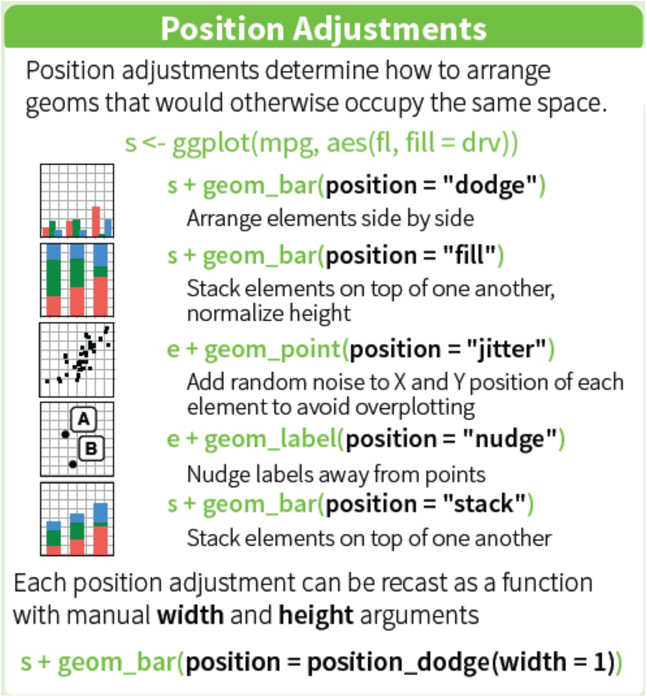
- Other GoG building blocks:
  - statistical transformations (“stat”) -- data summaries: mean, sd, binning & counting, …
  - scales: legends, axes to allow reading data from a plot



A stat builds new variables to plot (e.g., count, prop).

Grammar of Graphics

- Other GoG building blocks:
  - position adjustments: jitter, dodge, stack, …
  - faceting: small multiples or conditioning to break a plot into subsets.



Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

```
s <- ggplot(mpg, aes(fl, fill = drv))
```

```
s + geom_bar(position = "dodge")
```
Arrange elements side by side

```
s + geom_bar(position = "fill")
```
Stack elements on top of one another, normalize height

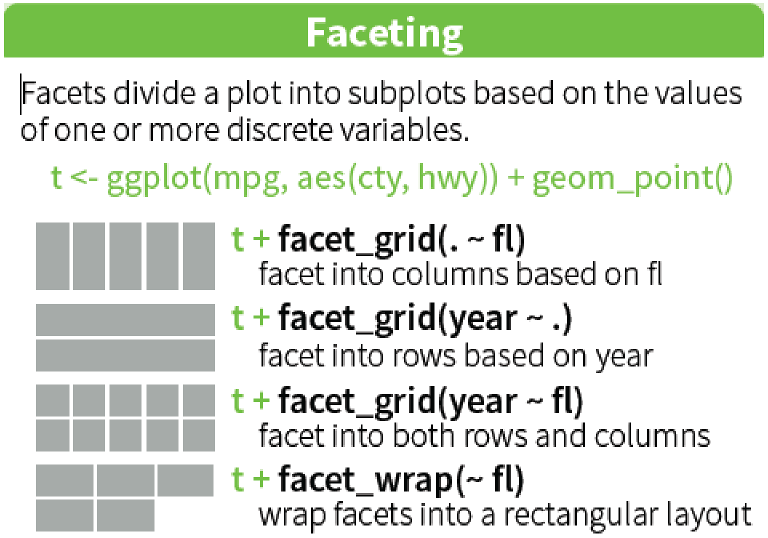
```
e + geom_point(position = "jitter")
```
Add random noise to X and Y position of each element to avoid overplotting

```
e + geom_label(position = "nudge")
```
Nudge labels away from points

```
s + geom_bar(position = "stack")
```
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments

```
s + geom_bar(position = position_dodge(width = 1))
```



Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

```
t <- ggplot(mpg, aes(cty, hwy)) + geom_point()
```

```
t + facet_grid(. ~ fl)
```
facet into columns based on fl

```
t + facet_grid(year ~ .)
```
facet into rows based on year

```
t + facet_grid(year ~ fl)
```
facet into both rows and columns

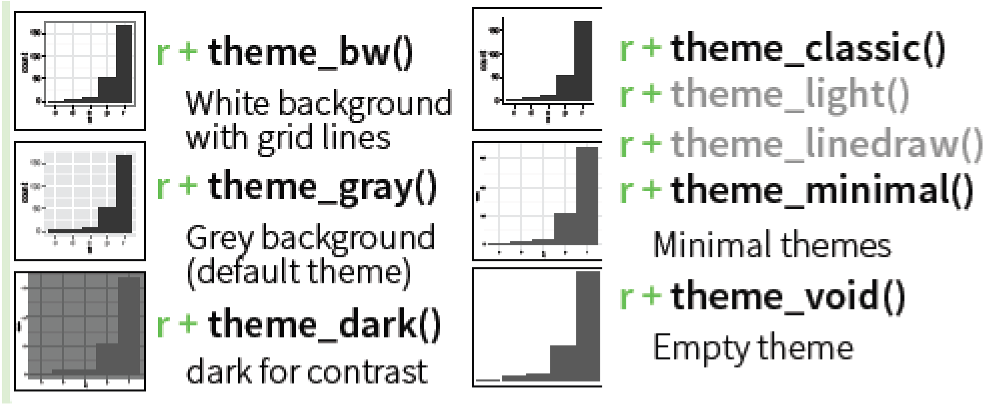
```
t + facet_wrap(~ fl)
```
wrap facets into a rectangular layout

ggplot2: GoG -> graphic language

- The implementation of GoG ideas in ggplot2 for R created a more expressive language for data graphs
  - layers: graph elements combined with "+" (read: "and")

```
ggplot(mtcars, aes(x=hp,  y=mpg)) +
    geom_point(aes(color = cyl)) +
    geom_smooth(method ="lm") +
```

  - themes: change graphic elements consistently



r + theme_bw()
White background with grid lines

r + theme_gray()
Grey background (default theme)

r + theme_dark()
dark for contrast

r + theme_classic()
r + theme_light()
r + theme_linedraw()
r + theme_minimal()
Minimal themes

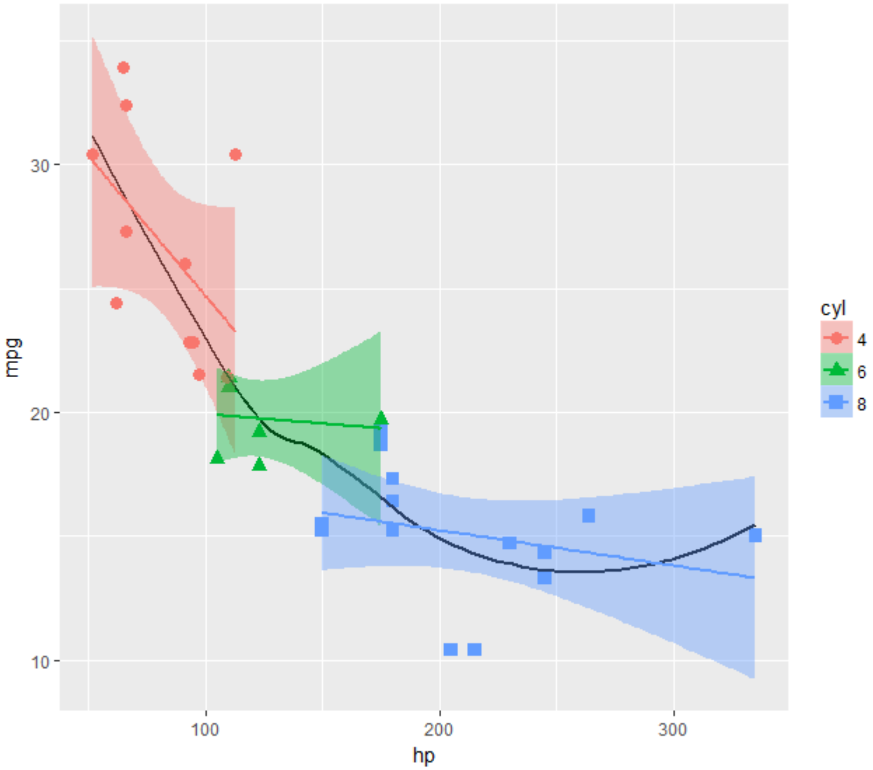
r + theme_void()
Empty theme

ggplot2: layers & aes()

Aesthetic attributes in the ggplot() call are passed to geom_() layers

Other attributes can be passed as **constants** (size=3, color="black") or with aes(color=, …) in different layers

This plot adds an overall loess smooth to the previous plot.
color="black" overrides the aes(color=cyl)



```
ggplot(mtcars, aes(x=hp, y=mpg)) +
   geom_point(size=3, aes(color=cyl, shape=cyl)) +
   geom_smooth(method="lm", aes(color=cyl, fill=cyl)) +
   geom_smooth(method="loess", color="black", se=FALSE)
```

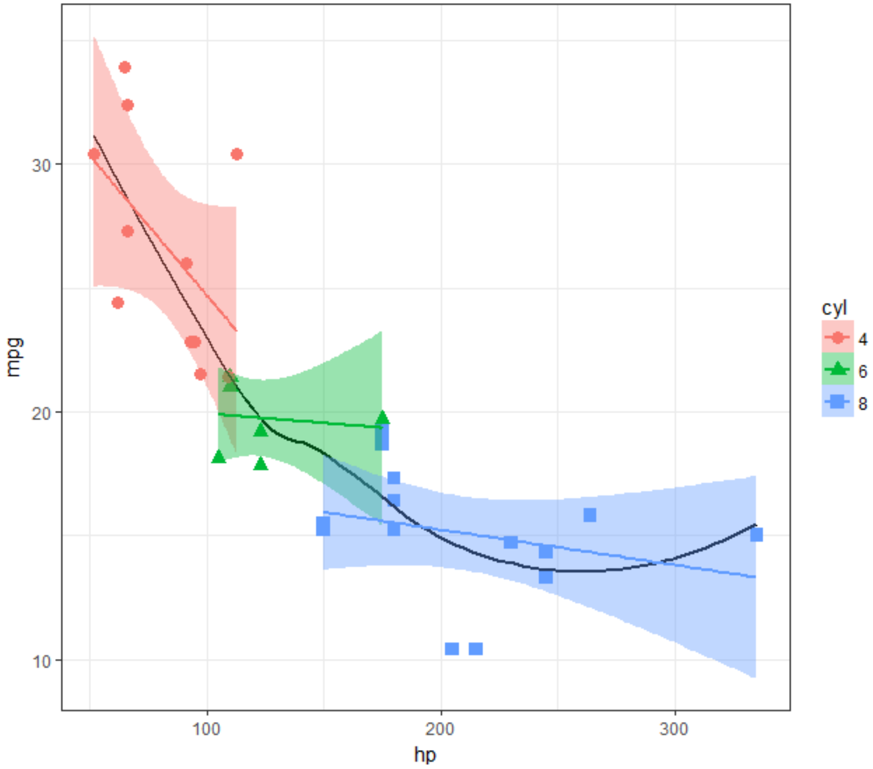
ggplot2: themes

All the graphical attributes of ggplot2 are governed by themes – settings for all aspects of a plot

A given plot can be rendered quite differently just by changing the theme

If you haven't saved the ggplot object, last_plot() gives you something to work with further

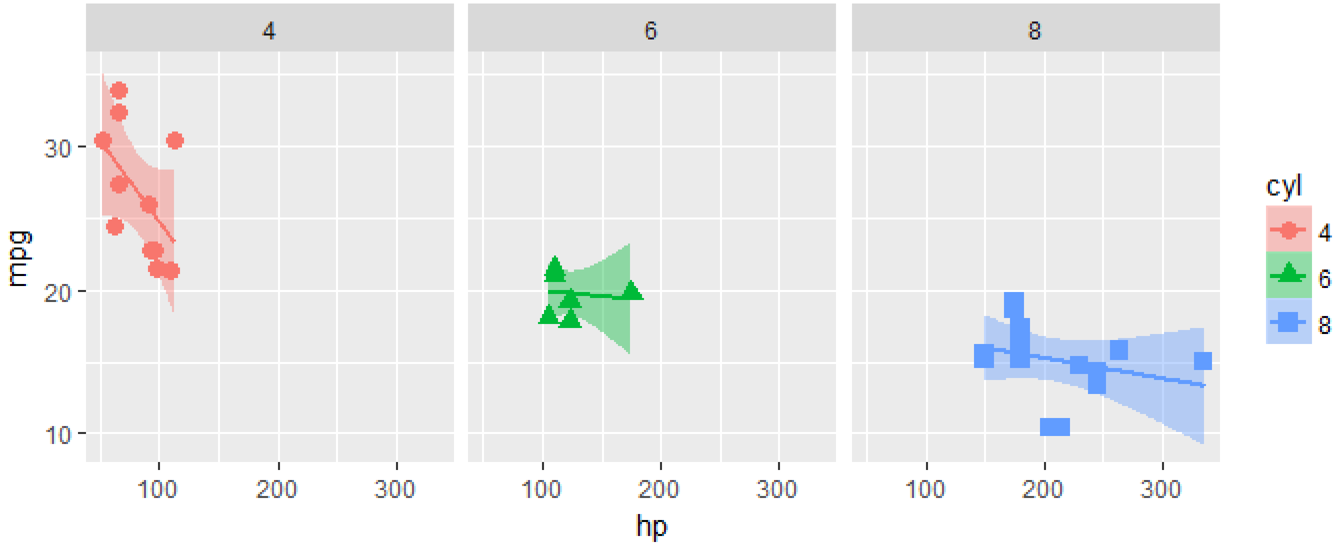
```
last_plot() + theme_bw()
```



ggplot2: facets

Facets divide a plot into separate subplots based on one or more discrete variables

```
plt <-
ggplot(mtcars, aes(x=hp, y=mpg, color=cyl, shape=cyl)) +
            geom_point(size=3) +
            geom_smooth(method="lm", aes(fill=cyl))

plt + facet_wrap(~cyl)
```



labeling points: geom_text()

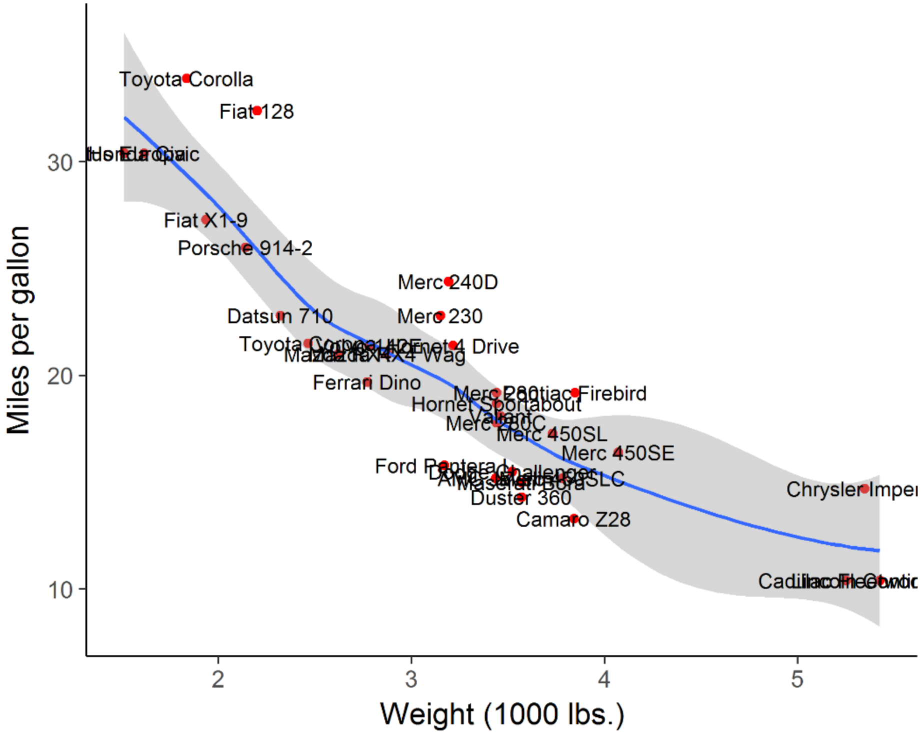
```
plt2 <- ggplot(mtcars, aes(x=wt, y=mpg)) +
   geom_point(color = 'red', size=2) +
   geom_smooth(method="loess") +
   labs(y="Miles per gallon", x="Weight (1000 lbs.)") +
   theme_classic(base_size = 16)

plt2 + geom_text(aes(label = rownames(mtcars)))
```

Sometimes it is useful to label points to show their identities.

geom_text() usually gives messy, overlapping text

Note the use of theme_classic() and better axis labels

But this is still messy: wouldn't want to publish this.

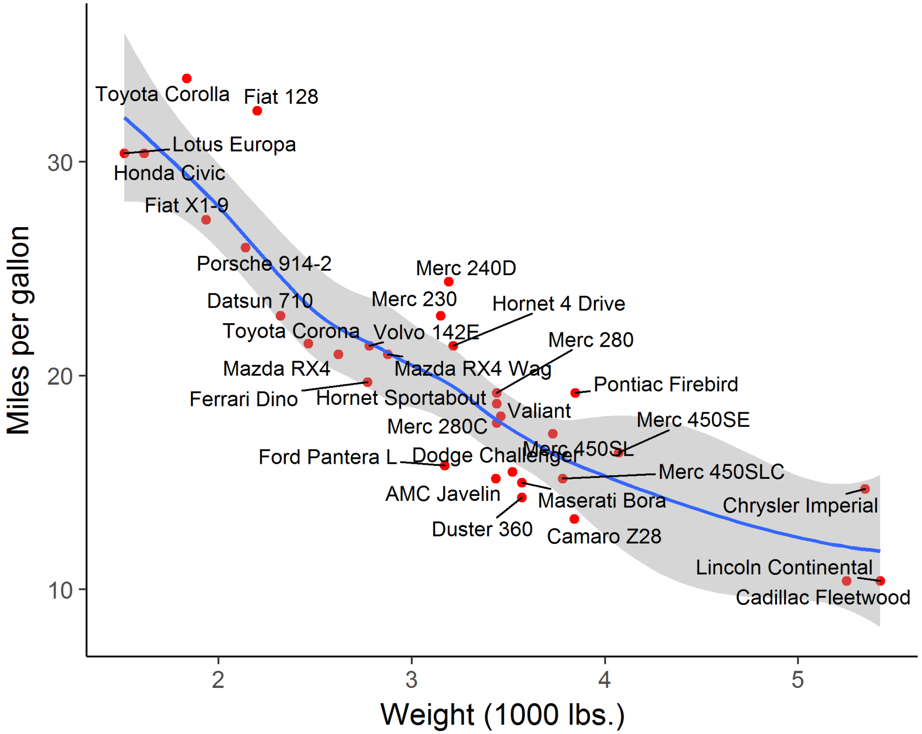


labeling points: geom_text_repel()

```
library(ggrepel)
plt2 +
   geom_text_repel(aes(label = rownames(mtcars)))
```

geom_text_repel() in the ggrepel package assigns repulsive forces among points and labels to assure no overlap

Some lines are drawn to make the assignment clearer



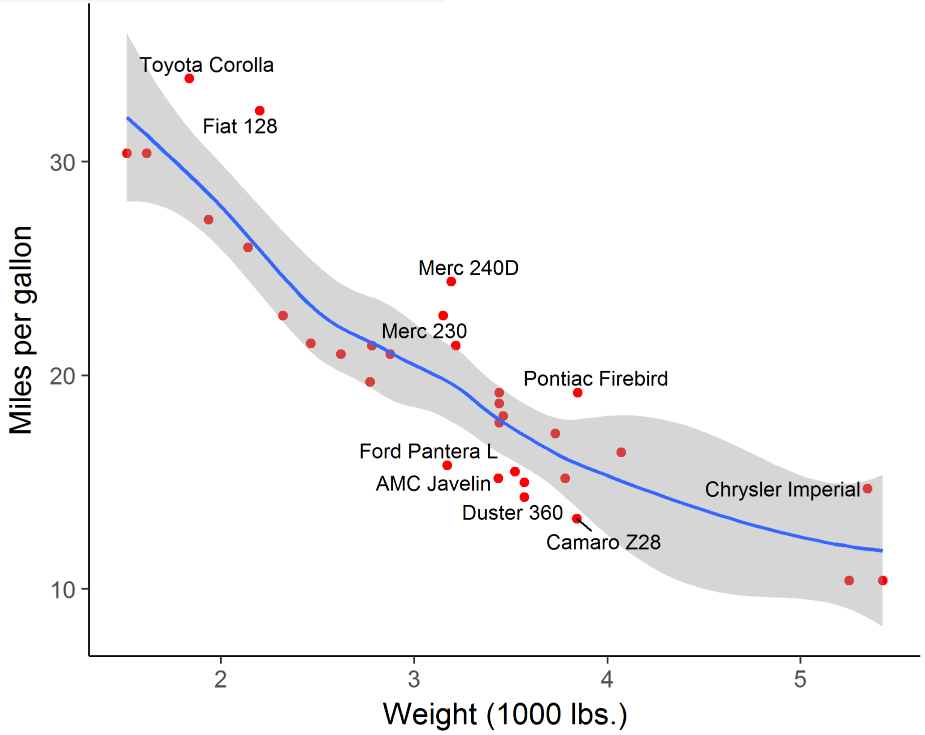
labeling points: selection

It is easy to label points selectively, using some criterion to assign labels to points

```
mod <- loess( mpg ~ wt, data=mtcars)                                    ❶
resids <- residuals(mod)                                                ❷
mtcars$label <- ifelse(abs(resids) > 2.5,                               ❸
                       rownames(mtcars), "")

plt2 + geom_text_repel(aes(label = mtcars$label))                       ❹
```

Here, I:

1. fit the smoothed loess curve,
2. extract residuals, r_i
3. assign labels where $|r_i| > 2.5$
4. add the text layer

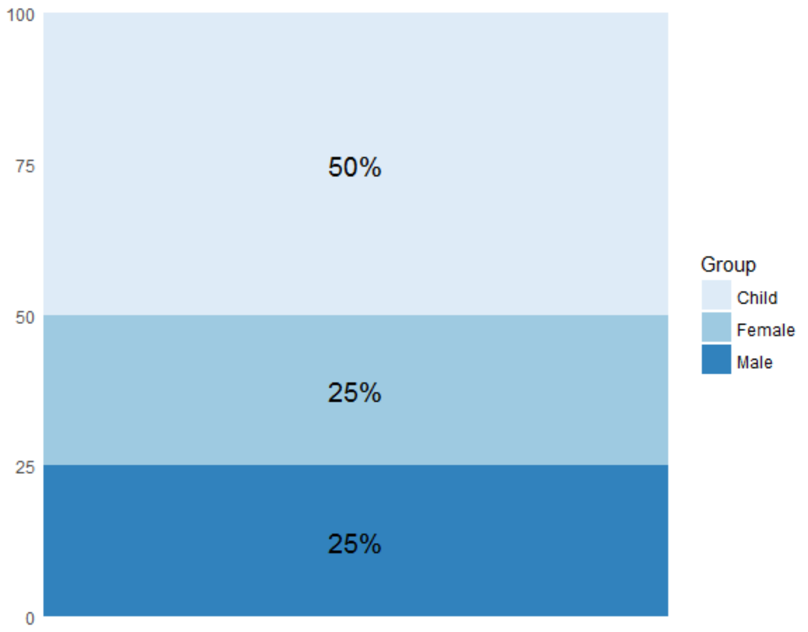


ggplot2: coords

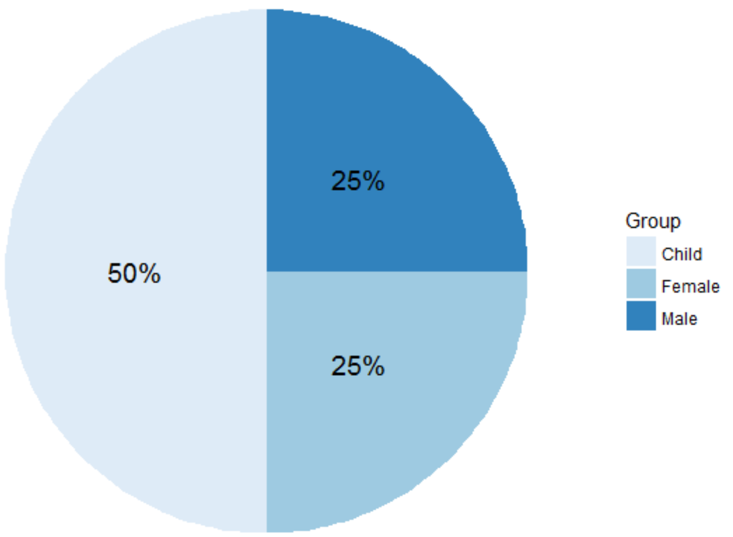
Coordinate systems, coord_*() functions, handle conversion from geometric objects to what you see on a 2D plot.

- A simple bar chart, standard coordinates
- A pie chart is just a bar chart in polar coordinates!

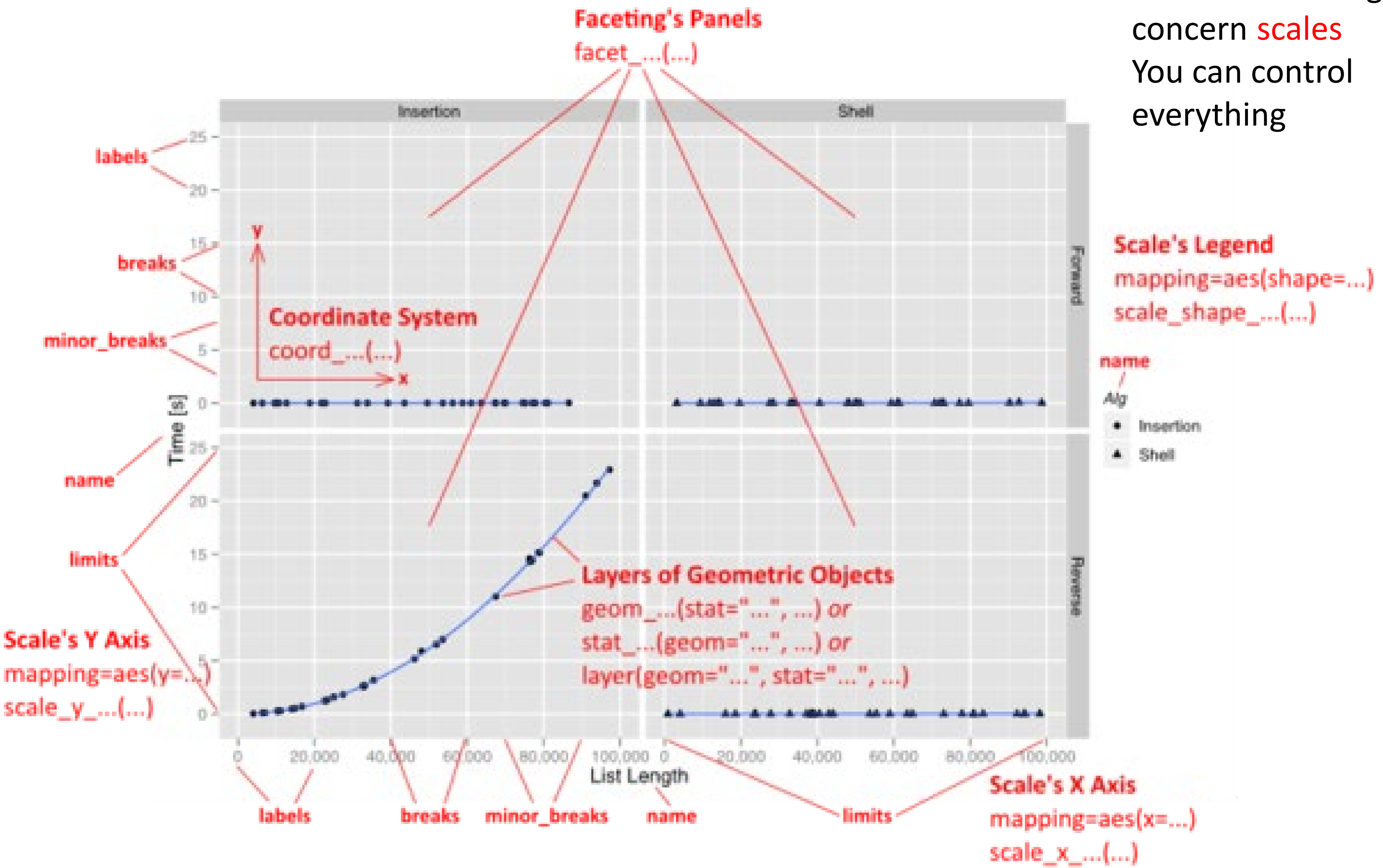
```
p <- ggplot(df, aes(x = "", y = value, fill = group)) +
        geom_bar( stat = "identity")
```



```
p + coord_polar("y", start = 0)
```



Anatomy of a ggplot



Other details of ggplot concern scales

You can control everything

ggplot objects

Traditional R graphics just produce graphical output on a device

However, ggplot() produces a “ggplot” object, a list of elements

```
> names(plt)
[1] "data"       "layers"     "scales"     "mapping"    "theme"      "coordinates"
[7] "facet"      "plot_env"   "labels"
> class(plt)
[1] "gg"     "ggplot"
```

What methods are available?

```
> methods(class="gg")
[1] +

> methods(class="ggplot")
[1] grid.draw    plot      print    summary
```

The “gg” class provides the “+” method

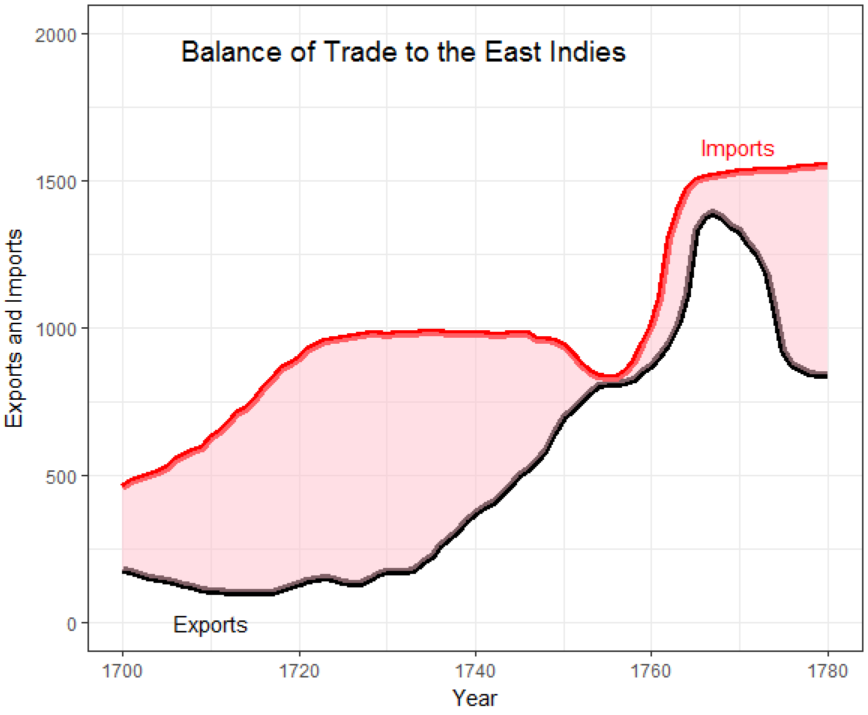
The “ggplot” class provides other, standard methods

Playfair: Balance of trade charts

In the *Commercial and Political Atlas*, William Playfair used charts of imports and exports from England to its trading partners to ask “How are we doing”?

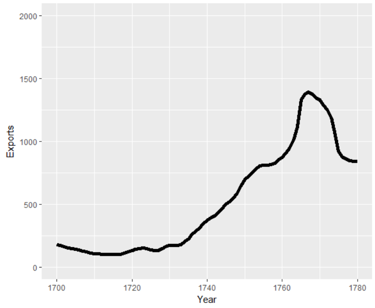
Here is a re-creation of one example, using ggplot2. How was it done?

```
> data(EastIndiesTrade,package="GDAdata")
> head(EastIndiesTrade)
  Year Exports Imports
1 1700     180     460
2 1701     170     480
3 1702     160     490
4 1703     150     500
5 1704     145     510
6 1705     140     525
   …        …       …
```



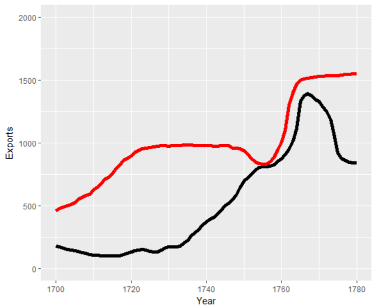
ggplot thinking

I want to plot two time series, & fill the area between them



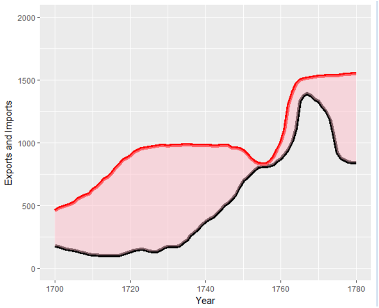
Start with a line plot of Exports vs. Year: **geom_line()**
Add a layer for the line plot of Imports vs. Year

```
c1 <-
ggplot(EastIndiesTrade, aes(x=Year, y=Exports)) +
        ylim(0,2000) +
        geom_line(colour="black", size=2) +
        geom_line(aes(x=Year, y=Imports), colour="red", size=2)
```



Fill the area between the curves: **geom_ribbon()**
change the Y label

```
c1 <- c1 +
   geom_ribbon(aes(ymin=Exports, ymax=Imports), fill="pink") +
   ylab("Exports and Imports")
```

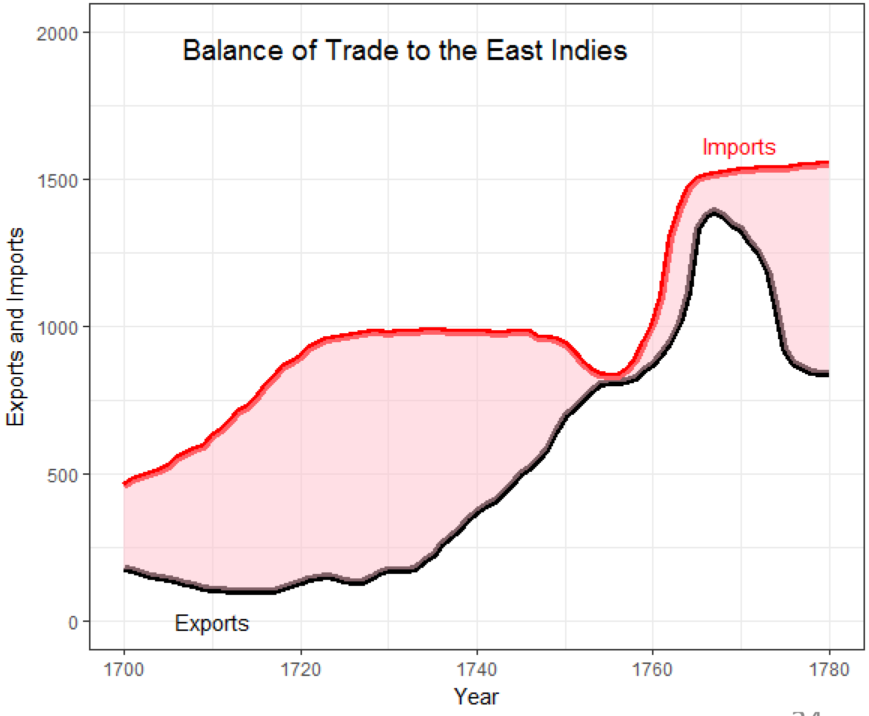


That looks pretty good. Add some text labels using **`annotate()`**

```
c1 <- c1 +
     annotate("text", x = 1710, y = 0, label = "Exports", size=4) +
     annotate("text", x = 1770, y = 1620, label = "Imports", color="red", size=4) +
     annotate("text", x = 1732, y = 1950, label = "Balance of Trade to the East Indies", color="black", size=5)
```

Finally, change the theme to b/w

```
c1 <- c1 + theme_bw()
```

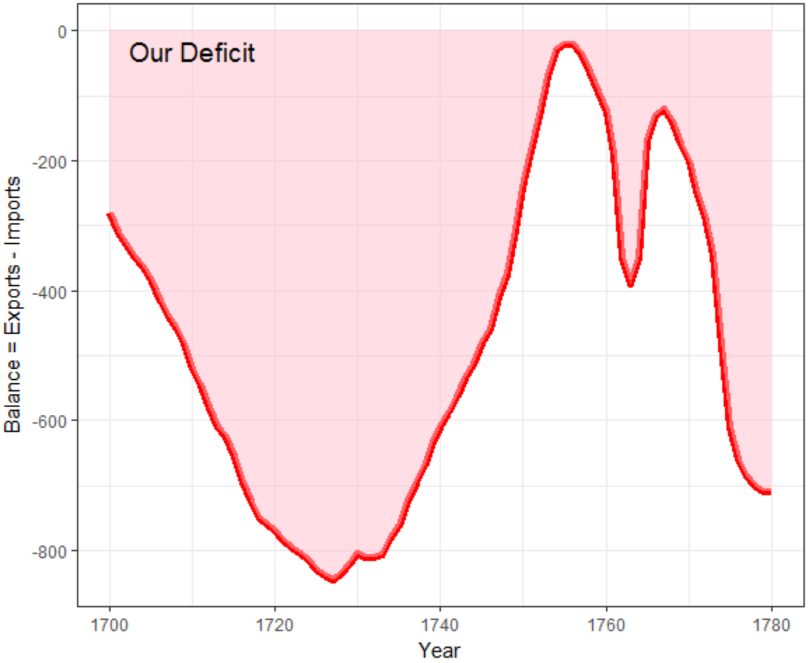


Plot what you want to show

Playfair's goal was to show the balance of trade with different countries.
Why not plot Exports – Imports directly?

```
c2 <-
ggplot(EastIndiesTrade, aes(x = Year, y = Exports - Imports)) +
    geom_line(colour="red", size=2) +
     ylab("Balance = Exports - Imports") +
     geom_ribbon(aes(ymin=Exports-Imports, ymax=0), fill="pink",alpha=0.5) +
     annotate("text", x = 1710, y = -30, label = "Our Deficit", color="black", size=5) +
     theme_bw()
```

aes(x=, y=) can use expressions calculated from data variables

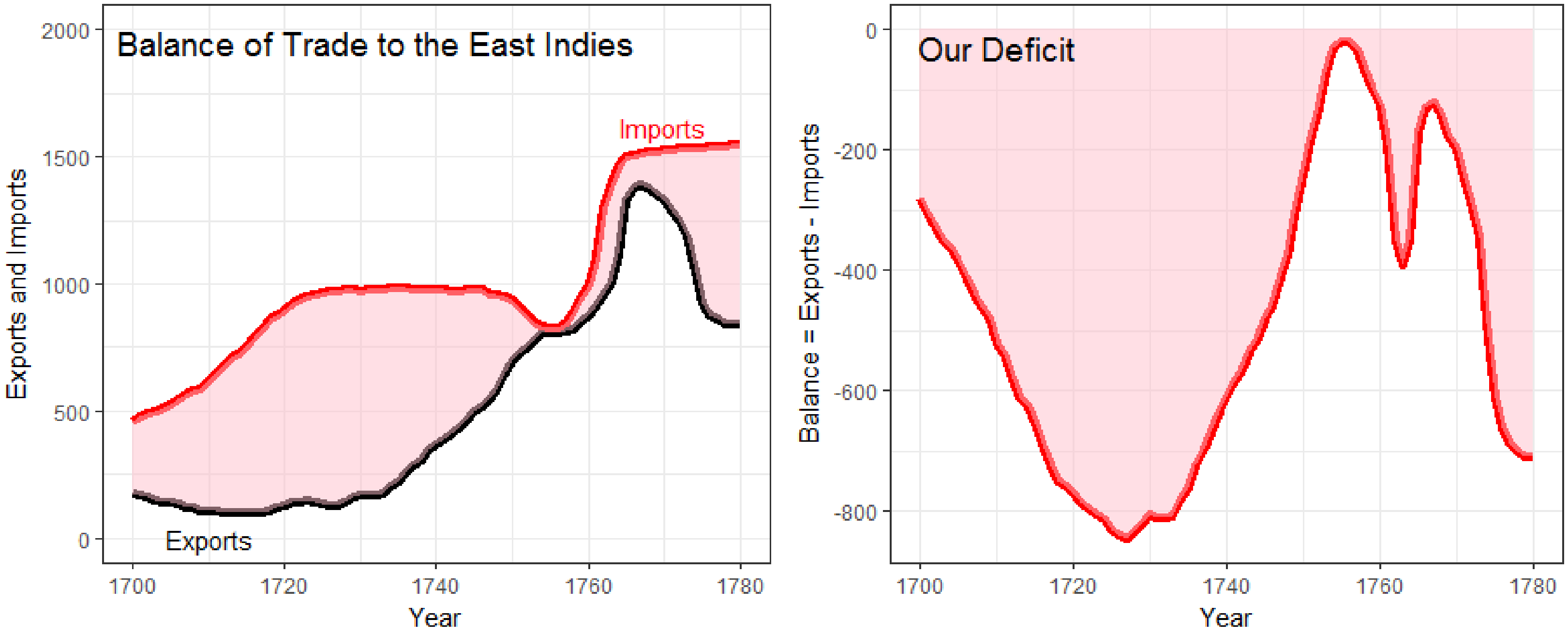


Composing several plots

ggplot objects use grid graphics for rendering

The gridExtra package has functions for combining or manipulating grid-based graphs

```
library(gridExtra)
grid.arrange(c1, c2, nrow=1)
```



Saving plots: ggsave()

- If the plot is on the screen

```
ggsave(“path/filename.png”)  # height=, width=
```

- If you have a plot object

```
ggsave(myplot, file=“path/filename.png”)
```

- Specify size:

```
ggsave(myplot, “path/filename.png”, width=6, height=4)
```

- any plot format (pdf, png, eps, svg, jpg, …)

```
ggsave(myplot, file=“path/filename.jpg”)
ggsave(myplot, file=“path/filename.pdf”)
```

Building a custom graph

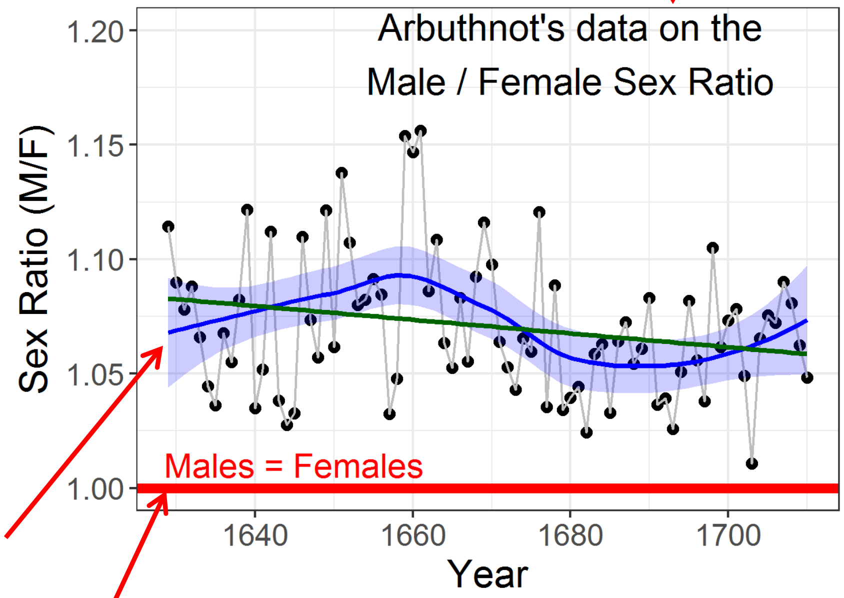
Custom graphs can be constructed by adding graphical elements (points, lines, text, arrows, etc.) to a basic ggplot()

John Arbuthnot: data on male/female sex ratios:

```
> data(Arbuthnot, package="HistData")
> head(Arbuthnot[,c(1:3,6,7)])
  Year Males Females Ratio Total
1 1629  5218    4683 1.114 9.901
2 1630  4858    4457 1.090 9.315
3 1631  4422    4102 1.078 8.524
4 1632  4994    4590 1.088 9.584
5 1633  5158    4839 1.066 9.997
6 1634  5035    4820 1.045 9.855
… …      …         …     …      …
```

Arbuthnot didn't make a graph. He simply calculated the probability that in 81 years from 1629—1710, the sex ratio would **always** be > 1
The first significance test!

figure caption



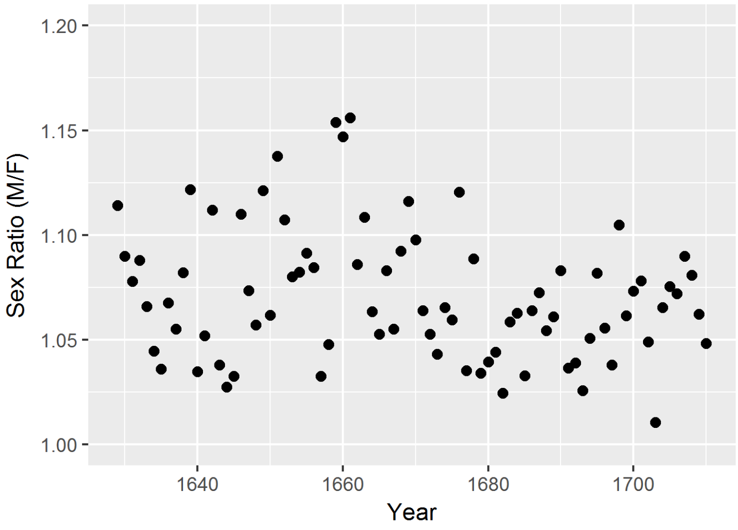
regression line & loess smooth

reference line

Building a custom graph

```
ggplot(Arbuthnot, aes(x=Year, y=Ratio)) +
   ylim(1, 1.20) +
   ylab("Sex Ratio (M/F)") +
   geom_point(pch=16, size=2)
```

Start with a basic scatterplot, Ratio vs. Year

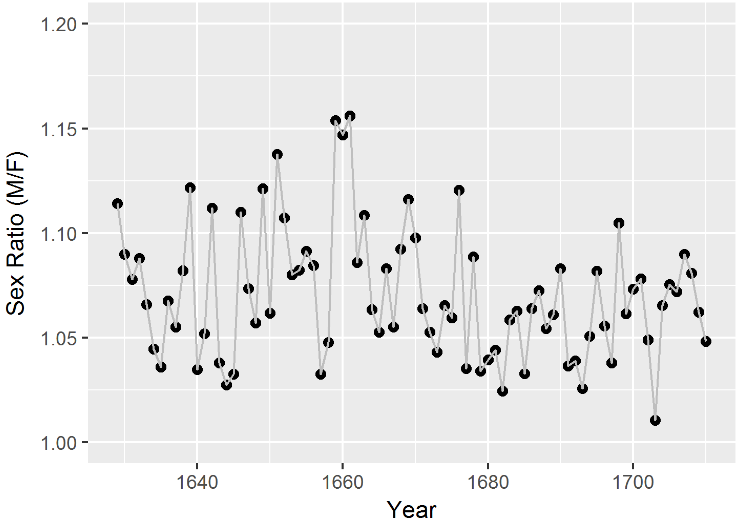


An R script for this example is available at:
http://euclid.psych.yorku.ca/www/psy6135/R/arbuthnot-gg.R

Building a custom graph

```
ggplot(Arbuthnot, aes(x=Year, y=Ratio)) +
   ylim(1, 1.20) +
   ylab("Sex Ratio (M/F)") +
   geom_point(pch=16, size=2) +
   geom_line(color="gray")
```

Connect points with a line

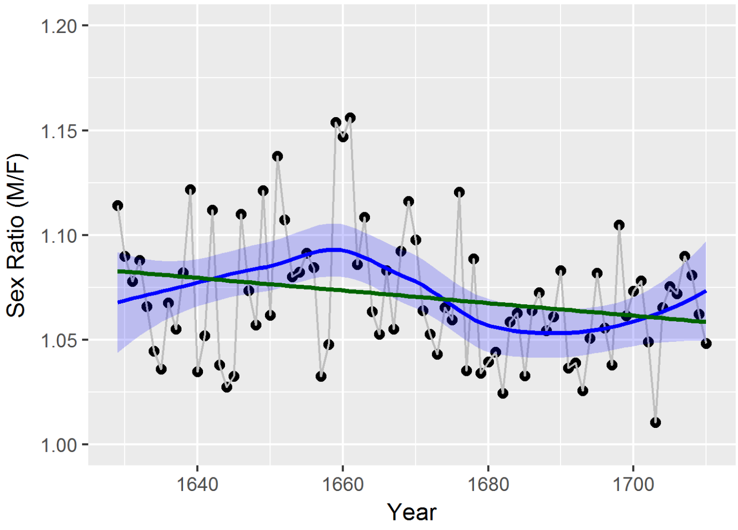


Building a custom graph

```
ggplot(Arbuthnot, aes(x=Year, y=Ratio)) +
   ylim(1, 1.20) +
   ylab("Sex Ratio (M/F)") +
   geom_point(pch=16, size=2) +
   geom_line(color="gray")  +
   geom_smooth(method="loess", color="blue",
          fill="blue", alpha=0.2) +
   geom_smooth(method="lm", color="darkgreen",
          se=FALSE)

# save what we have so far
arbuth <- last_plot()
```

Add smooths:

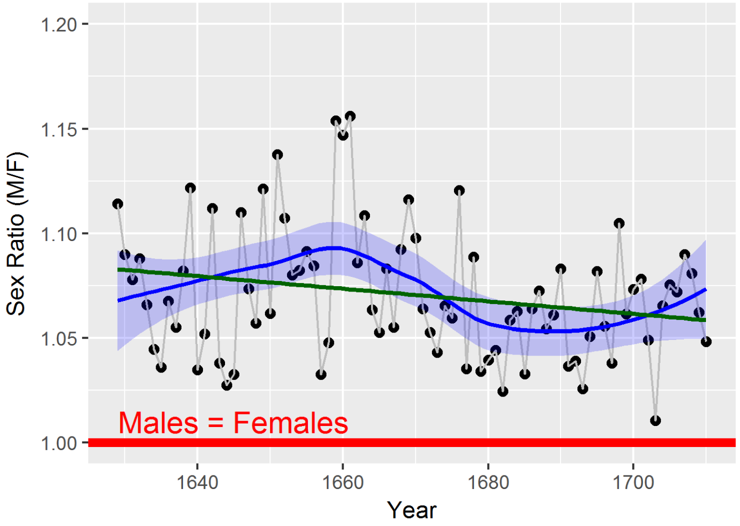
- loess curve
- linear regression line



Building a custom graph

```
arbuth +
    geom_hline(yintercept=1, color="red", size=2) +
    annotate("text", x=1645, y=1.01, label="Males = Females", color="red", size=5)
```

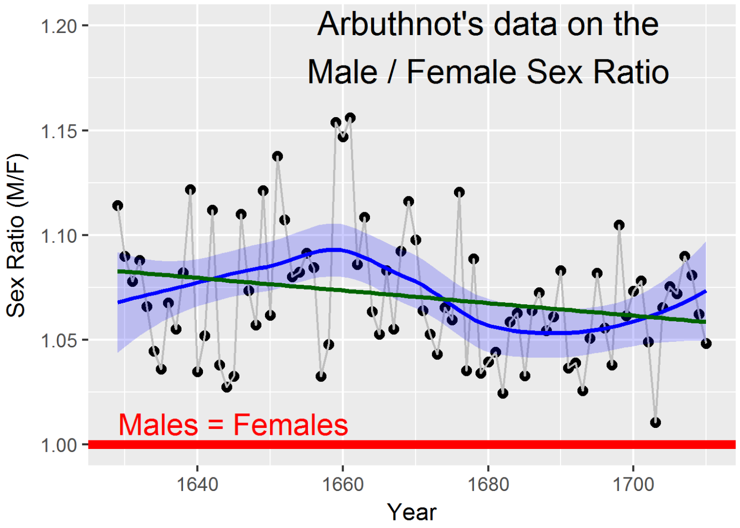
Add horizontal reference line & text label



Building a custom graph

```
arbuth +
   geom_hline(yintercept=1, color="red", size=2) +
   annotate("text", x=1645, y=1.01, label="Males = Females", color="red", size=5) +
   annotate("text", x=1680, y=1.19,
          label="Arbuthnot's data on the\nMale / Female Sex Ratio", size=5.5)
```

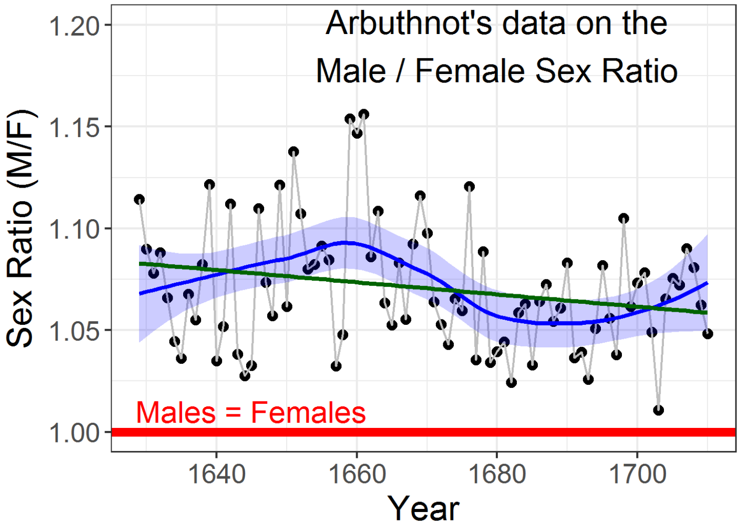
Add figure title



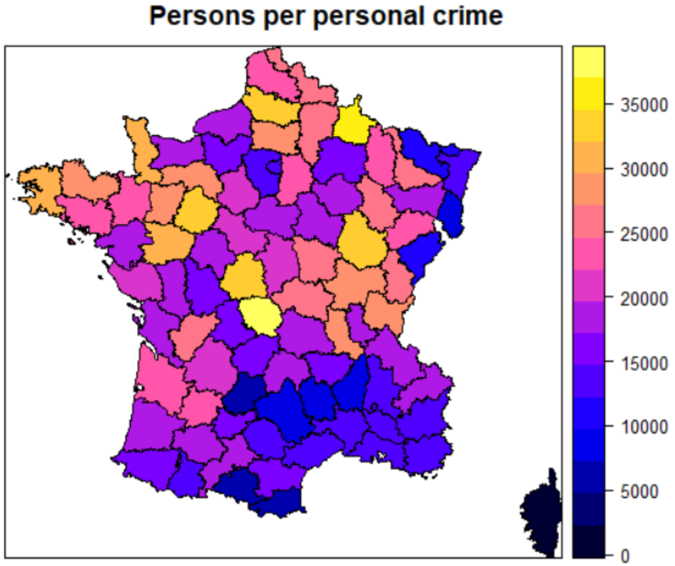
Building a custom graph

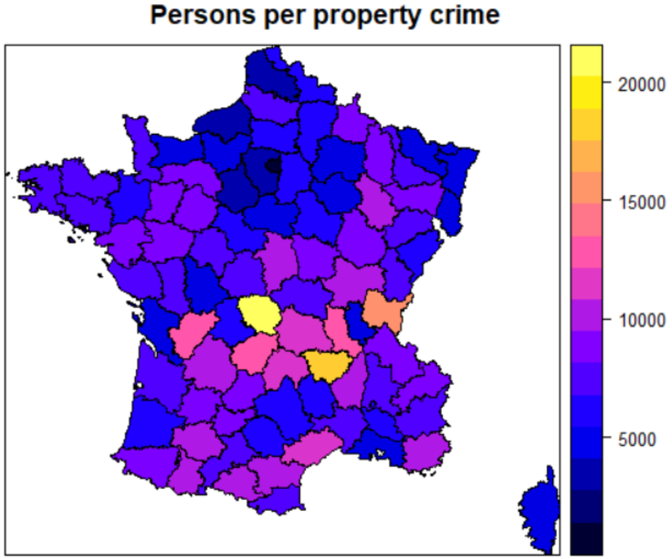
```
arbuth +
   geom_hline(yintercept=1, color="red", size=2) +
   annotate("text", x=1645, y=1.01, label="Males = Females", color="red", size=5) +
   annotate("text", x=1680, y=1.19,
          label="Arbuthnot's data on the\nMale / Female Sex Ratio", size=5.5) +
   theme_bw() + theme(text = element_text(size = 16))
```

Change the theme and font size



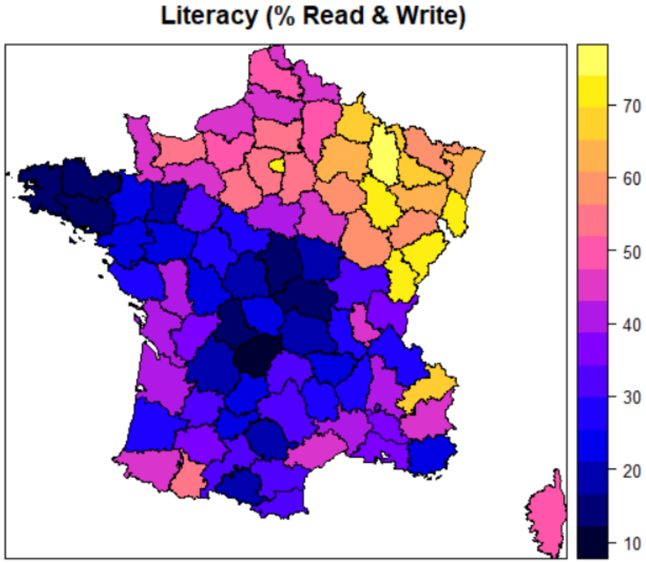
Guerry: Moral statistics of France





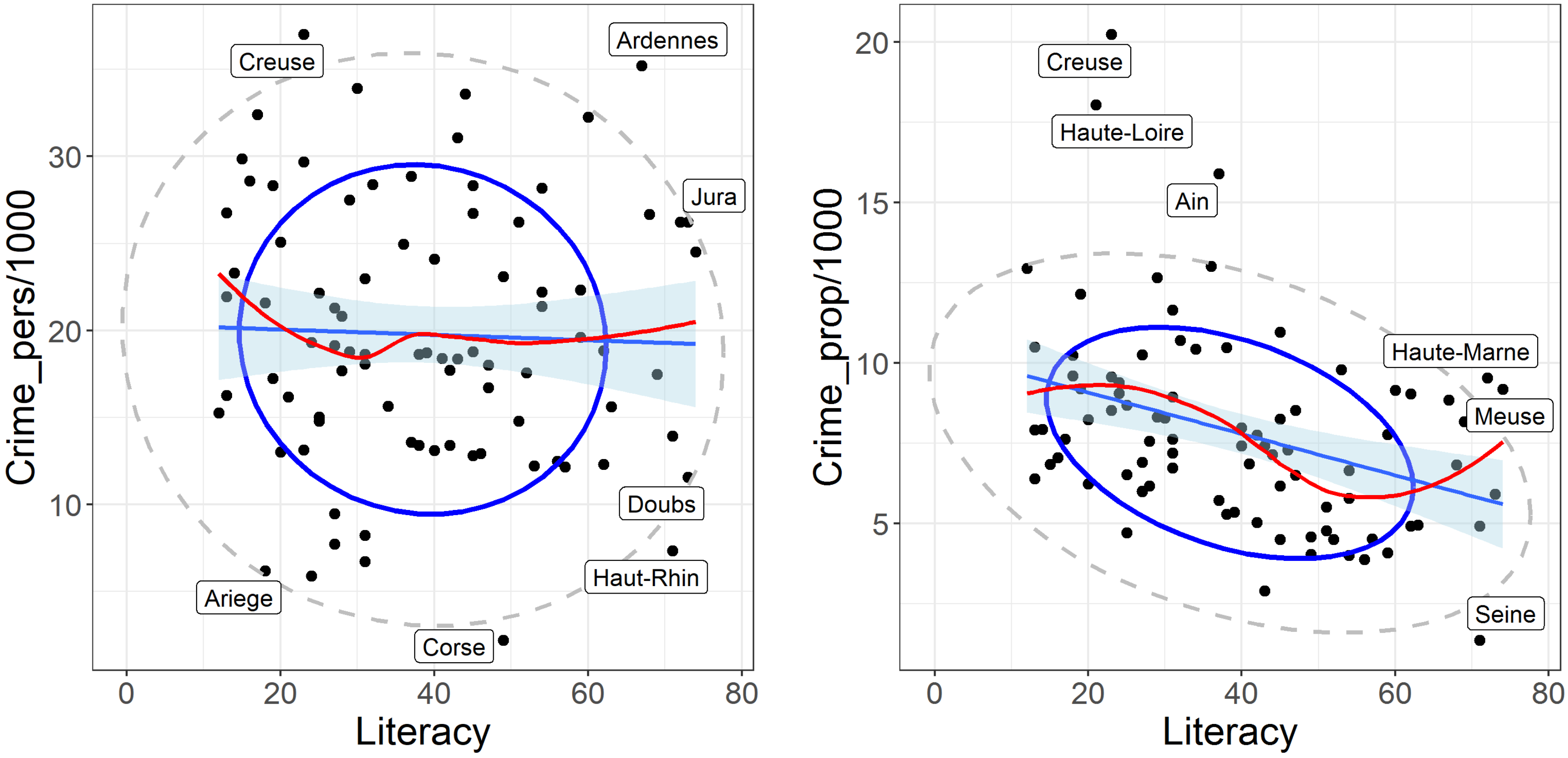
Guerry (1833) made shaded maps of France to determine if crime was related to literacy & other factors

```
library(Guerry)
library(sp)
spplot(gfrance, "Crime_pers")
spplot(gfrance, "Crime_prop")
spplot(gfrance, "Literacy")
```



Consulting for Guerry

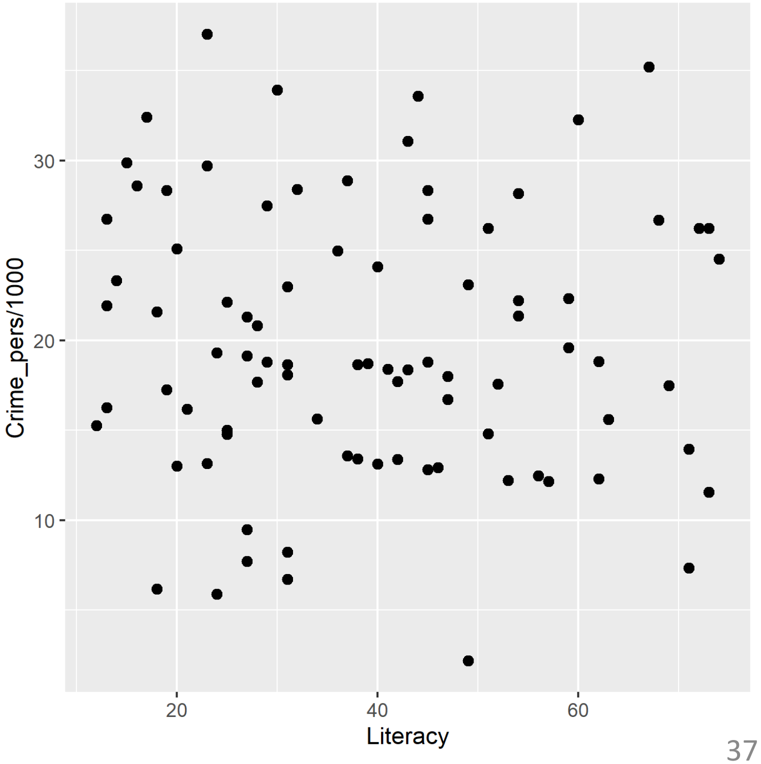
Guerry: Mes cartes sont très jolies, non? But how can I go further?
MF: Make scatterplots! Add smooths & data ellipses. See you next week at Café Lillas
Guerry: Les boissons sont sur moi!



Building Guerry’s plots

```
ggplot(aes(x=Literacy, y=Crime_pers/1000), data=Guerry) +
  geom_point(size=2)
```

Start with a basic scatterplot

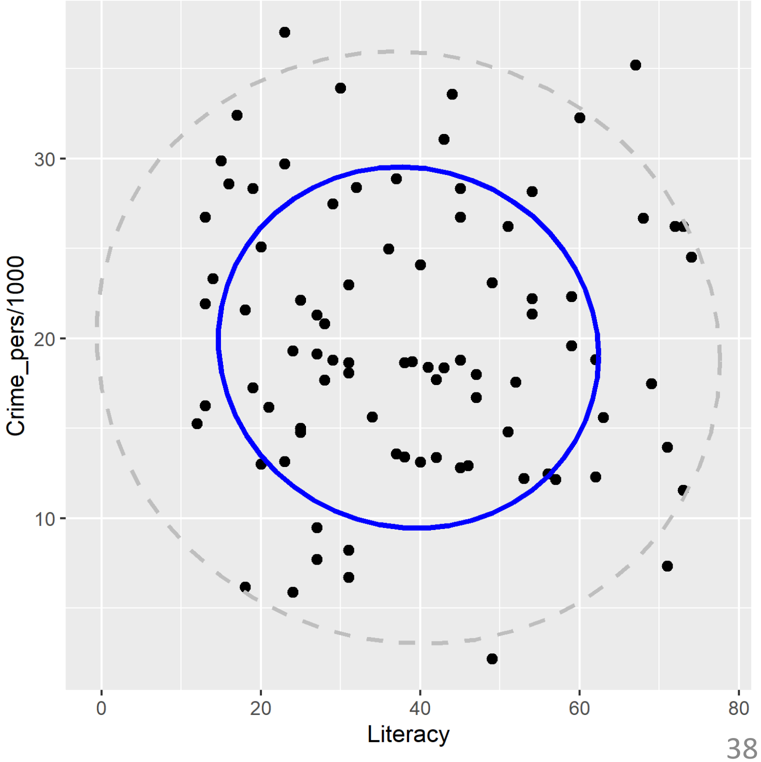


Building Guerry’s plots

```
ggplot(aes(x=Literacy, y=Crime_pers/1000), data=Guerry) +
  geom_point(size=2) +
  stat_ellipse(level=0.68, color="blue", size=1.2) +
  stat_ellipse(level=0.95, color="gray", size=1, linetype=2)
```

Add data ellipses to show correlation

- 68% ~ mean ± 1 sd
- 95% ~ mean ± 2 sd



Guerry’s plots: Add smooths

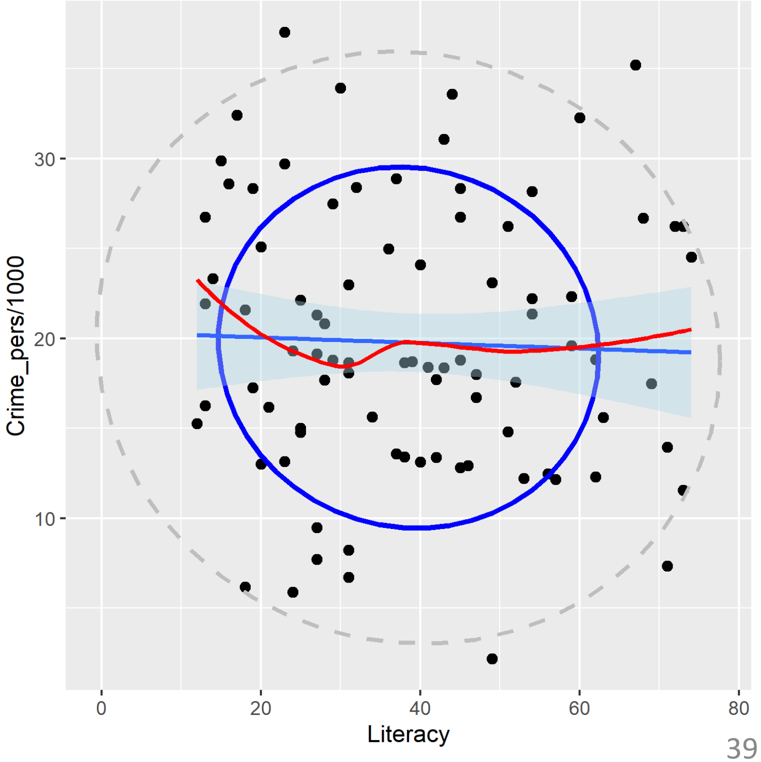
```
ggplot(aes(x=Literacy, y=Crime_pers/1000), data=Guerry) +
  geom_point(size=2) +
  stat_ellipse(level=0.68, color="blue", size=1.2) +
  stat_ellipse(level=0.95, color="gray", size=1, linetype=2) +
  geom_smooth(method="lm", formula=y~x, fill="lightblue") +
  geom_smooth(method="loess", formula=y~x, color="red", se=FALSE)
```

Add lm() and loess() smooths

- lm shows regression slope
- loess diagnoses possible non-linearity

Coffee break: save the current plot object

```
gplot <- last.plot()
```



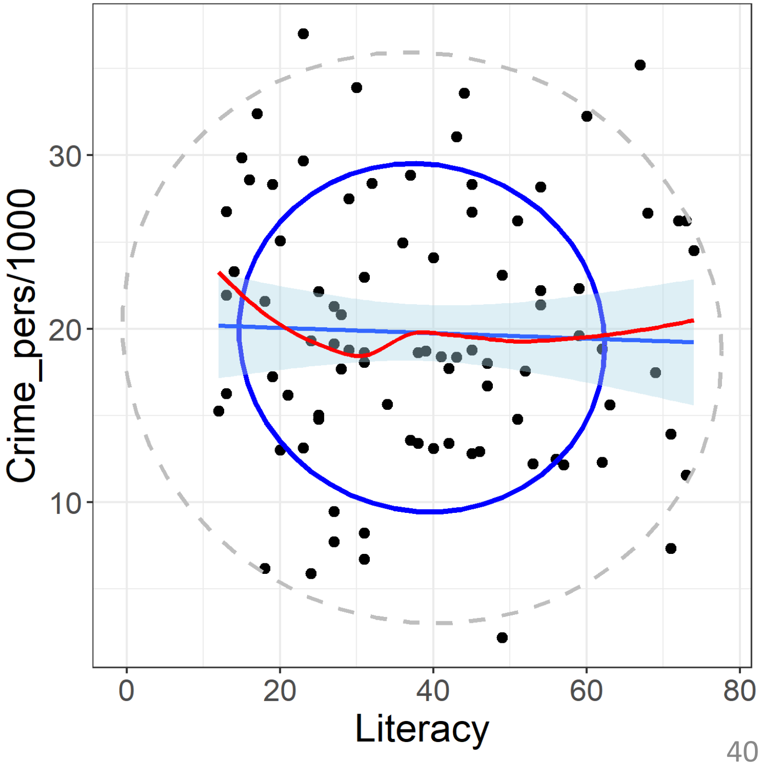
Guerry’s plots: Styling

Guerry: I want to publish this! But need to make axis labels larger

```
gplot <- last_plot()
gplot + theme_bw() +
      theme(text = element_text(size=18))
```

MF:

- Change the basic theme to theme_bw()
- Increase the font size for all text
- You can change the style of anything you want



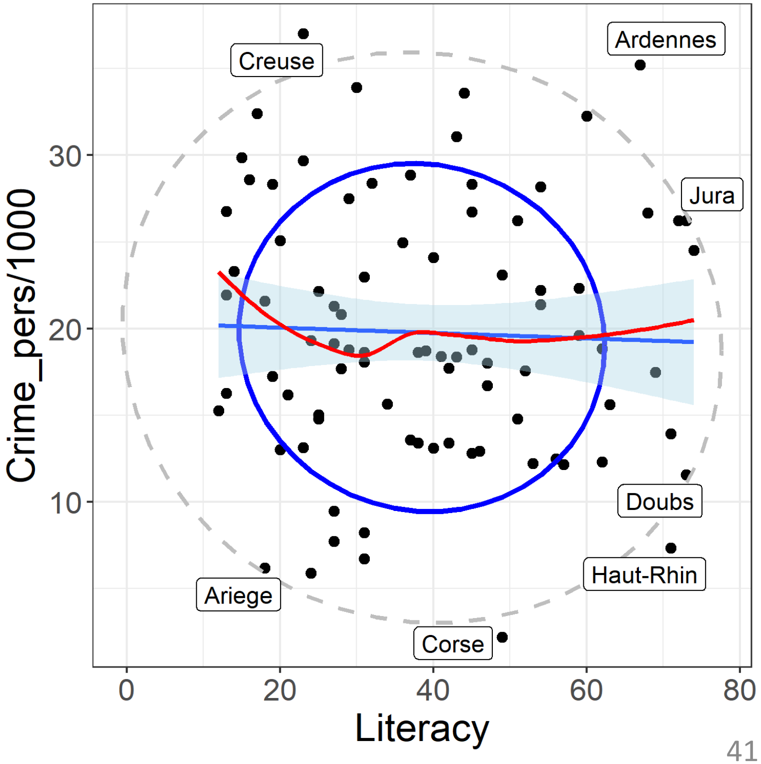
Guerry's plots: Labeling

Guerry: OK, but I see some unusual points. What are they?
MF: Need to calculate "unusualness" – Mahalanobis D^2 squared distance from centroid

```
gdf <- Guerry[, c("Literacy", "Crime_pers", "Department")]
gdf$dsq <- mahalanobis(gdf[,1:2], colMeans(gdf[,1:2]), cov(gdf[,1:2]))
```

$$D^2 = (x - \bar{x})' S^{-1} (x - \bar{x})'$$

```
library(ggrepel)
gplot +
    theme_bw() +
    theme(text = element_text(size=18)) +
    geom_label_repel(aes(label=Department),
            data = gdf[gdf$dsq > 4.6,])
```


Creuse
Ardennes
Jura
Doubs
Haut-Rhin
Ariege
Corse
Crime_pers/1000
Literacy