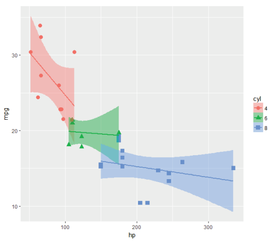
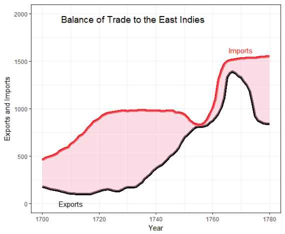
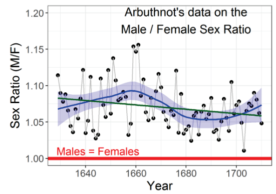






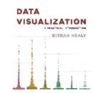
Introduction to ggplot2

Michael Friendly
Psych 6135

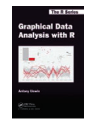
https://friendly.github.io/6135

Resources: Books

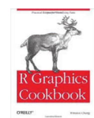
Hadley Wickham, *ggplot2: Elegant graphics for data analysis*, 2nd Ed.
1st Ed: Online, http://ggplot2.org/book/
ggplot2 Quick Reference: http://r-statistics.co/ggplot2-cheatsheet.html
Complete ggplot2 documentation: http://docs.ggplot2.org/current/



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.



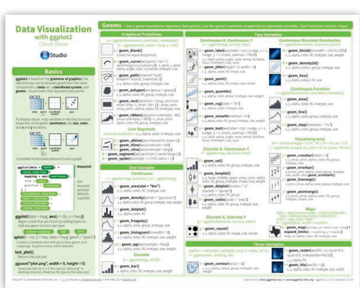
Antony Unwin, *Graphical Data Analysis with R*
A gentile introduction to doing visual data analysis, mainly with ggplot2.
R code: http://www.gradaanwr.net/

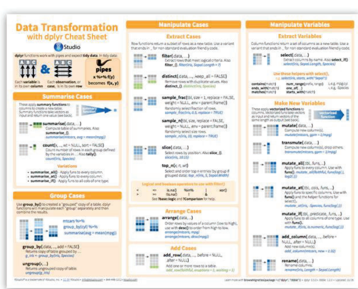


Winston Chang, *R Graphics Cookbook: Practical Recipes for Visualizing Data*
Cookbook format, covering common graphing tasks; the main focus is on ggplot2
R code from book: http://www.cookbook-r.com/Graphs/
Download from: http://ase.tufts.edu/bugs/guide/assets/R%20Graphics%20Cookbook.pdf

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





Topics

- ggplot overview
  - components: geoms, stats, scales, …
  - getting stylish: themes
- Some examples
  - Playfair, balance of trade
  - Arbuthnot’s birth ratios
  - Minard’s enhanced scatterplots
- Beyond 2D
- Animation

What is ggplot2?

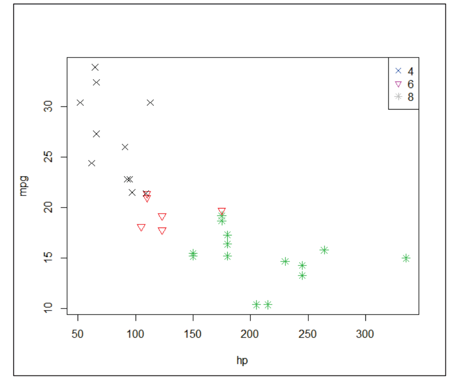
- ggplot2 is Hadley Wickham's R package for producing "elegant graphics for data analysis"
  - An implementation of the ideas for graphics introduced in Lee Wilkinson's *Grammar of Graphics*
  - These ideas and the syntax of ggplot2 help to think of graphs in a new and more general way
  - Produces pleasing 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/

ggplot 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"

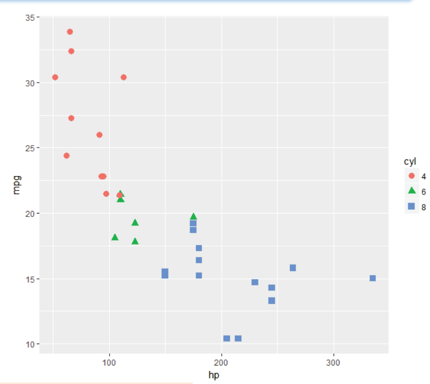
ggplot vs base graphics

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

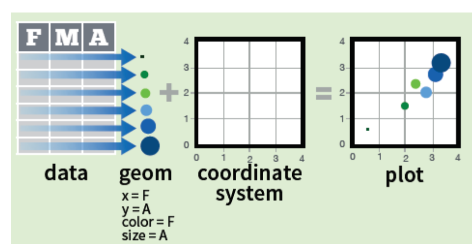



```
library(ggplot2)
ggplot(mtcars, aes(x=hp, y=mpg, color=cyl, shape=cyl)) +
   geom_point(size=3)
```

Follow along: the R script for this example is at: https://friendly.github.io/6135/R/gg-cars.R

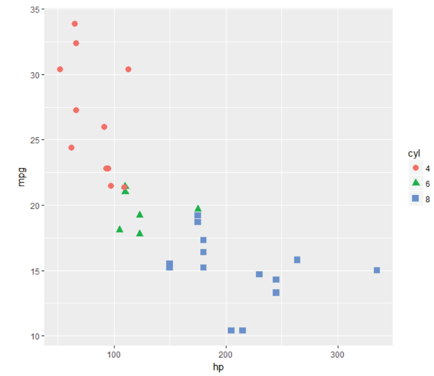
ggplot components

- 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,



ggplot: data + geom -> graph

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



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)

ggplot geoms: basic

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

ggplot geoms: 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

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

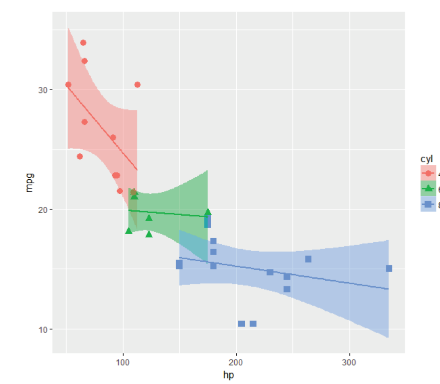
h + **geom_hex()**
x, y, alpha, colour, fill, size

ggplot: geoms

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!)



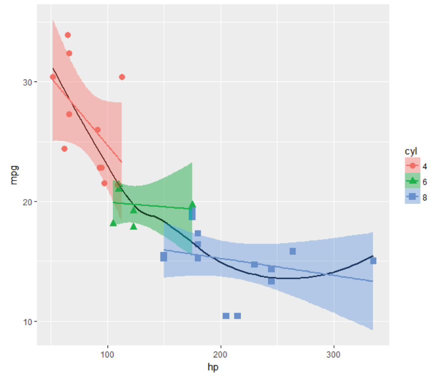

```
ggplot(mtcars, aes(x=hp, y=mpg, color=cyl, shape=cyl)) +
  geom_point(size=3) +
  geom_smooth(method="lm", aes(fill=cyl))
```

ggplot: 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

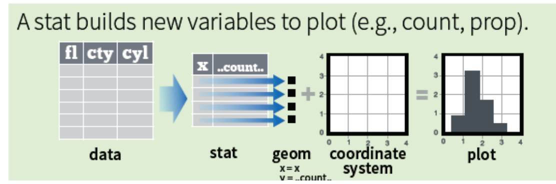
This plot adds an overall loess smooth to the previous plot.
color="black" overrides the aes(color=cyl)




```
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)
```


ggplot: stats

- statistical calculations ("stat") -- data summaries: mean, sd, binning & counting, …
- stats have a default geom and geoms have a default stat
  - geom_bar(stat = "count") ← → stat_count(geom = "bar")
- computed variables (eg, ..count.., ..level..) can be mapped to aesthetics
  - ggplot(aes(x=, y=)) + stat_density_2d(aes(fill = ..level..), geom="polygon")



some stats:

- stat_count()
- stat_bin()
- stat_density()
- stat_boxplot()
- stat_density_2d()
- stat_ellipse()

scales

Scales map data values to the visual values of an aesthetic.

- axis labels, legends, colors
- formatting of values

General Purpose scales
Use with most aesthetics

scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times.
Use same arguments as scale_x_date().
See ?strptime for label formats.

X and Y location scales
Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

Color and fill scales (Discrete)
n <- d + geom_bar(aes(fill = fl))

n + **scale_fill_brewer**(palette = "Blues")
For palette choices: RColorBrewer::display.brewer.all()
n + **scale_fill_grey**(start = 0.2, end = 0.8, na.value = "red")

Color and fill scales (Continuous)
o <- c + geom_dotplot(aes(fill = ..x..))

o + **scale_fill_distiller**(palette = "Blues")
o + **scale_fill_gradient**(low="red", high="yellow")
o + **scale_fill_gradient2**(low="red", high="blue", mid = "white", midpoint = 25)
o + **scale_fill_gradientn**(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

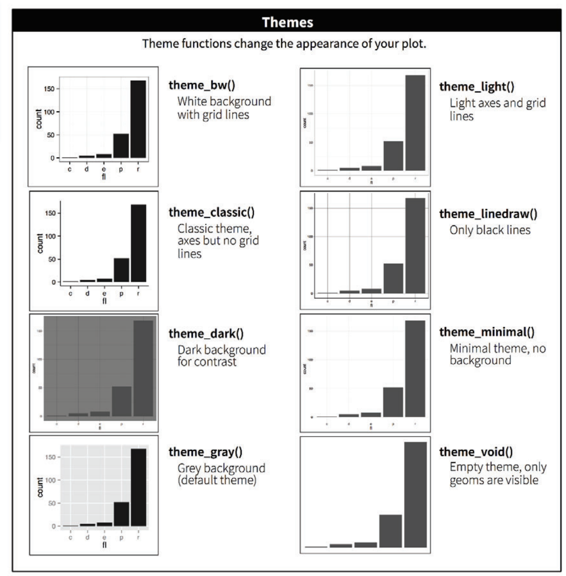
ggplot themes

ggplot themes set the general style for all graphic elements

Designed to provide a pleasing coherent design – simply!

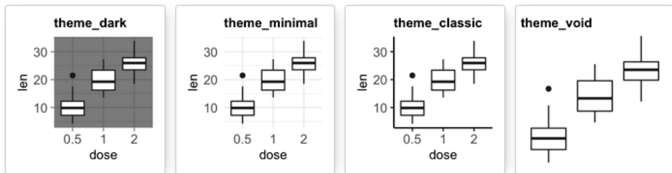
Yet, all the details can be customized

One theme can rule them all!

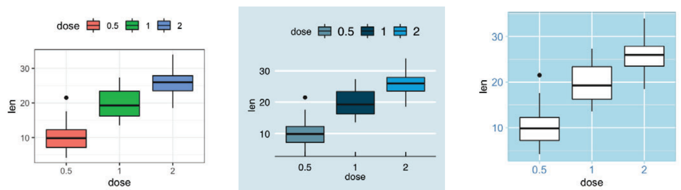


ggplot2: themes

Built-in ggplot themes provide a wide variety of basic graph styles



Other packages provide custom themes, or you can easily define your own



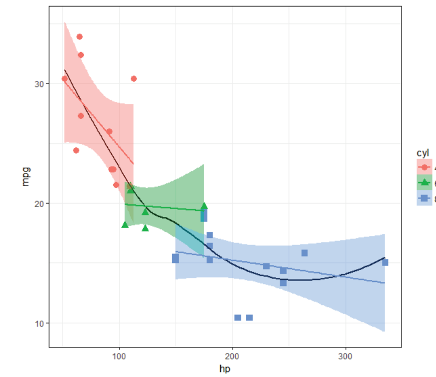
theme_hc() theme_economist() theme_bluewhite()

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()
```


Other ggplot features

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + coord_trans(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

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

Set **scales** to let axis limits vary across facets

t + facet_grid(drv ~ fl, **scales = "free"**)
x and y axis limits adjust to individual facets

- **"free_x"** - x axis limits adjust
- **"free_y"** - y axis limits adjust

Zoom / clip : use xlim, ylim = c(lo,hi)

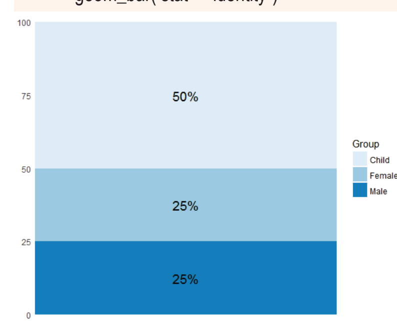
ggplot2: coords

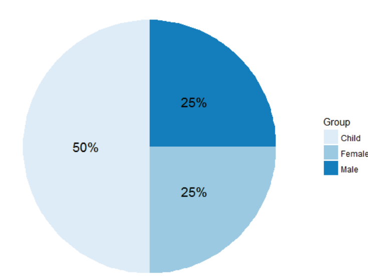
Coordinate systems, coord_*() functions, handle conversion from geometric objects to what you see on a 2D plot.

- 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)
```





labeling points: geom_text()

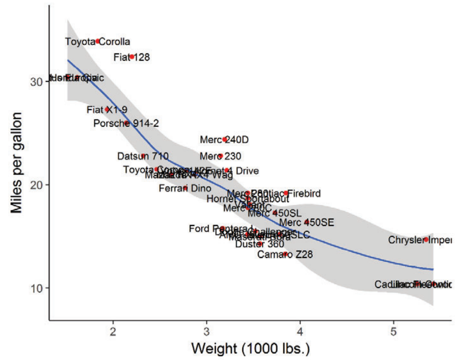
```
plt2 <- ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point(color = 'red', size=2) +
  geom_smooth(method="loess") +
  labs(y="Miles per gallon", x="Weight (1000 lbs.)") +
  theme_classic(base_size = 16)

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(), better axis labels and increased font size.

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

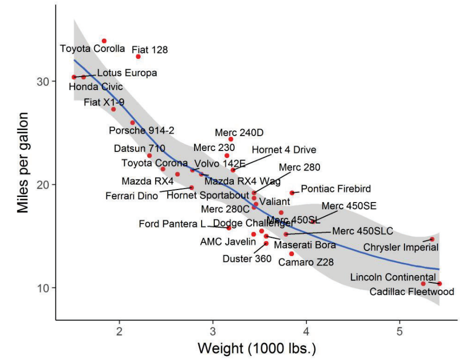


labeling points: geom_text_repel()

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

geom_text_repel() assigns repulsive forces among points and labels to assure no overlap

Some lines are drawn to make the assignment clearer



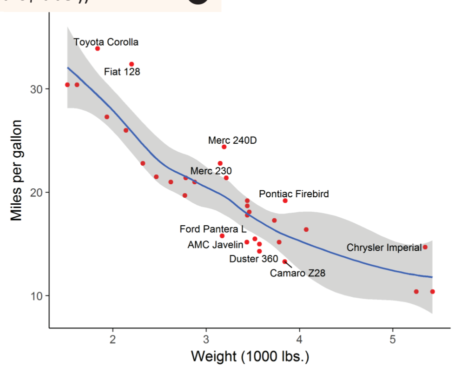
labeling points: selection

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



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

Saving plots: ggsave()

- If the plot is on the screen

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

- If you have a plot object

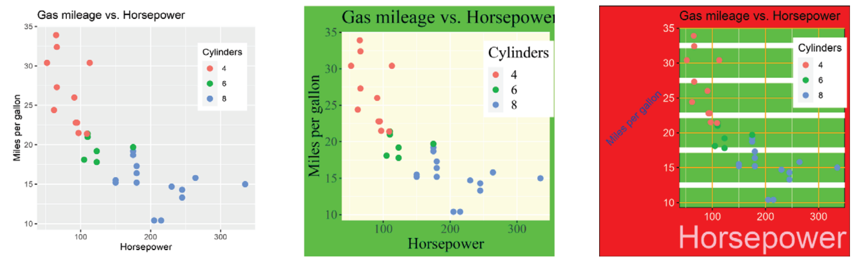
```
ggsave(myplot, file="path/filename.png")
```

- Specify size:

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

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

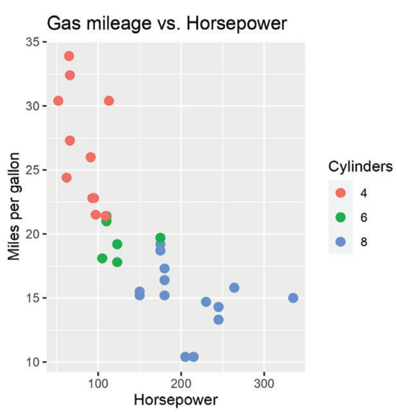
```
ggsave(myplot, file="path/filename.jpg")
ggsave(myplot, file="path/filename.pdf")
```

STYLING WITH THEMES

Options vs. themes

```
p1 <-
  ggplot(mtcars, aes(x = hp, y=mpg)) +
  geom_point(aes(color=factor(cyl)),
             size=3, pch=16 ) +
  labs(color = "Cylinders",
       x = "Horsepower",
       y = "Miles per gallon") +
  ggtitle("Gas mileage vs. Horsepower")

p1
```



Some graphic attributes (size, symbol) are controlled as options of geoms

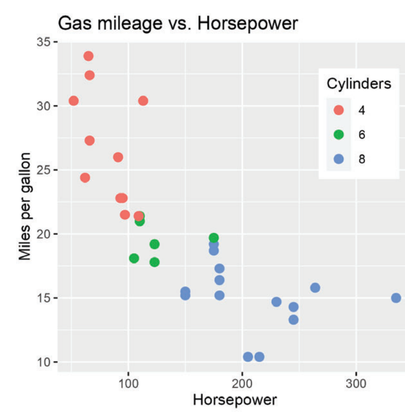
Others must be controlled by theme elements

ggplot default: theme_gray()

Moving the legend

Move the legend inside the plot region

```
p1 + theme(
      legend.position = c(0.85, 0.75))
```

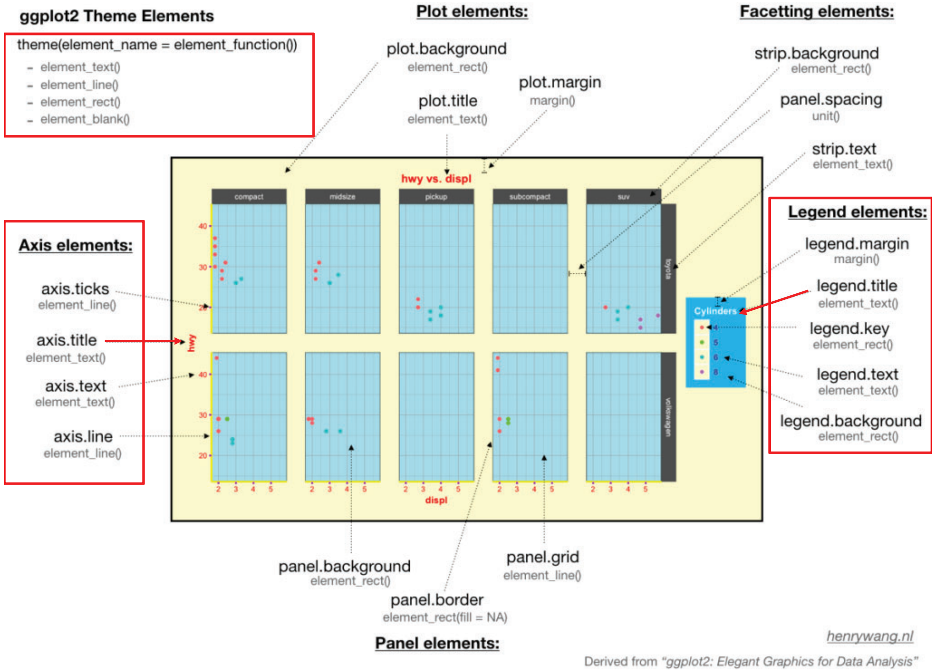


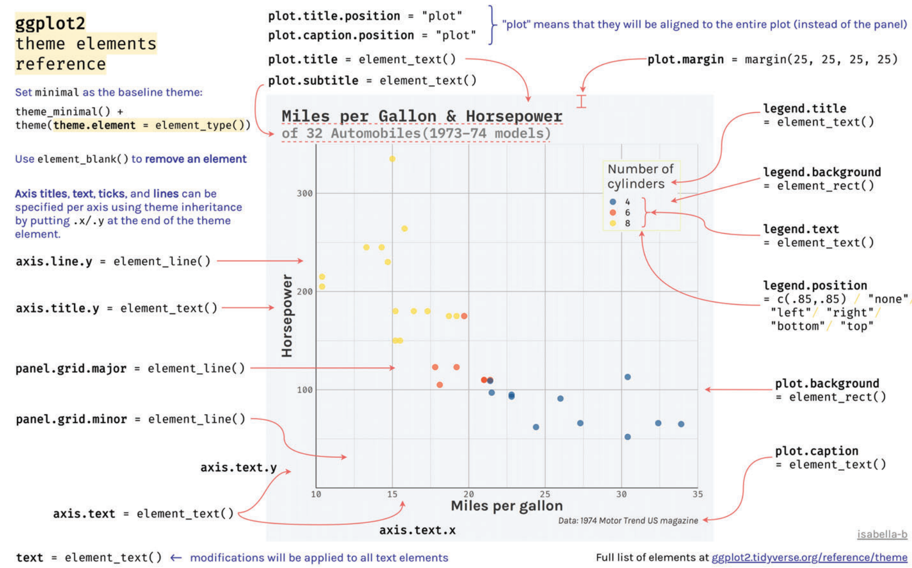
This gives more room for the data

Legend position is a theme attribute

Might also want to control:
- legend title: font, size, label…
- legend background, margin, …

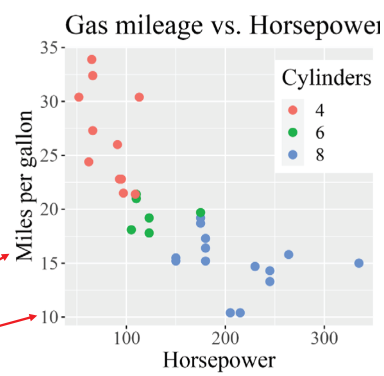
Need to know their names & properties





Styling with themes: text

```
p1 + theme(
        legend.position = c(0.85, 0.75),
        text = element_text(size=18, family = "serif") )
```



Change the font family & size for all text.

Change specific text items:

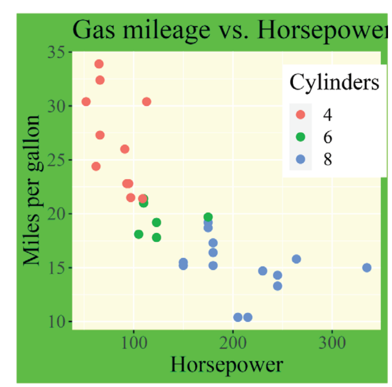
- axis.title = element_text()
- axis.text = element_text()
- legend.title =
- legend.text =
- ...

NB: could also use
theme_gray(base_size=, base_family=, ...)

Styling with themes: backgrounds

Change the backgrounds

```
p1 + theme(
        legend.position = c(0.85, 0.75),
        text = element_text(size=18, family = "serif"),
        plot.background = element_rect(fill="green"),
        panel.background =
                    element_rect(fill="lightyellow")
   )
```



Custom themes

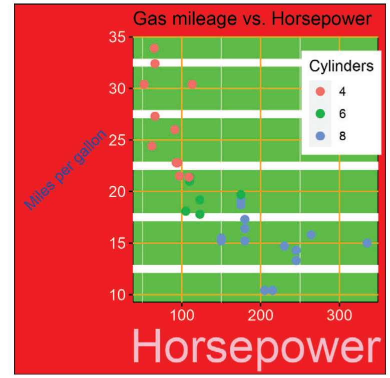
You can define & save your own theme and then use to style all plots consistently

```
theme_ugly <- function (base_size = 12, base_family = "") {
 theme_gray(base_size = base_size, base_family = base_family)      base theme
  %+replace%
   theme(
    axis.text = element_text(colour = "white"),
    axis.title.x = element_text(colour = "pink", size=rel(3)),
    axis.title.y = element_text(colour = "blue", angle=45),
    panel.background = element_rect(fill="green"),                   override
    panel.grid.minor.y = element_line(size=3),                       these
    panel.grid.major = element_line(colour = "orange"),              properties
    plot.background = element_rect(fill="red")
   )
}
```


Styling with themes

Use your theme:

```
p1 + theme_ugly() +
     theme(legend.position = c(0.85, 0.75))
```



To set a theme as the default for all subsequent graphics:

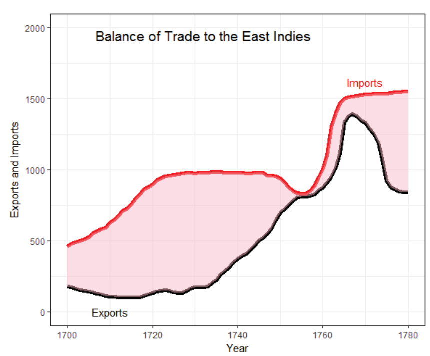
- theme_set(theme_bw())
- theme_set(theme_ugly()

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"?

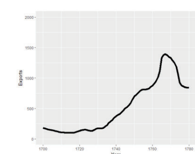
Here is a re-creation of one example, using ggplot2. How was it done?

```
> 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
   ...     ...     ...
```



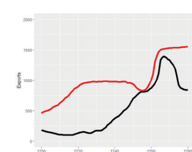
ggplot thinking

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



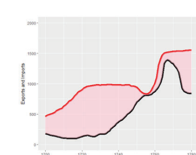
Start with a line plot of Exports vs. Year: **geom_line()**
Add a layer for the line plot of Imports vs. Year

```
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")
```

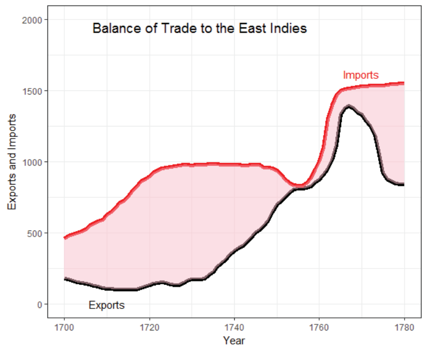


That looks pretty good. Add some text labels using **`annotate()`**

```
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()
```

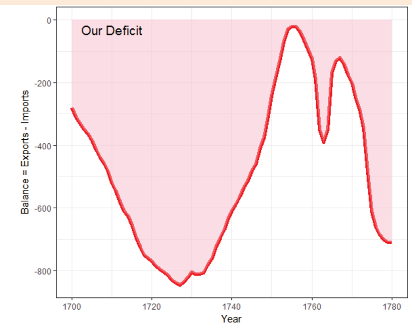


Plot what you want to show

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

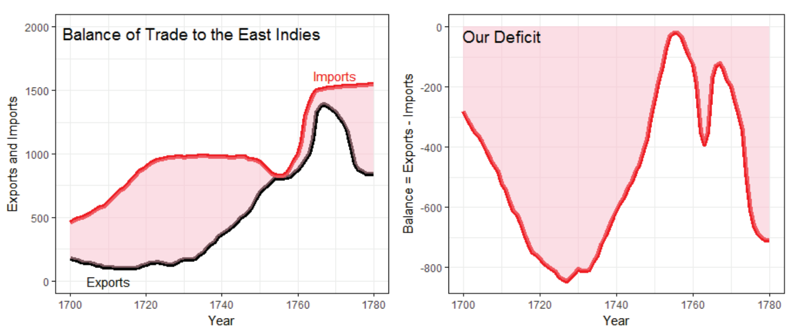


Composing several plots

ggplot objects use grid graphics for rendering

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

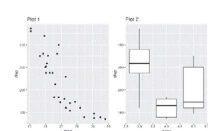
```
library(gridExtra)
grid.arrange(c1, c2, nrow=1)
```



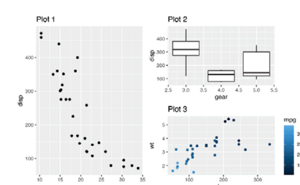
General composing: patchwork

The patchwork package provides a new, complete syntax for plot compositions

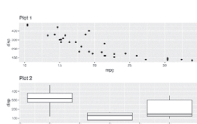
p1 + p2
side-by-side



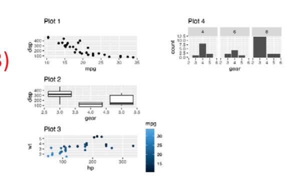
p1 + (p2 / p3)
grouping



p1 / p2
up/down



p1 + p2 +p3 + p4 +
plot_layout(nrow=3)
fancy layouts



Arbuthnot's data on birth ratios

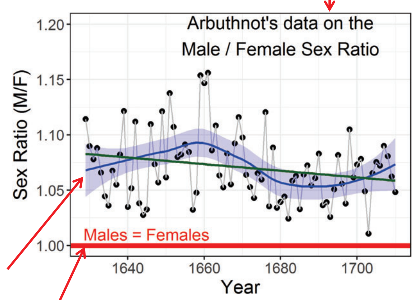
Custom graphs can be constructed by adding graphical elements (points, lines, text, arrows, etc.) to a basic ggplot()

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

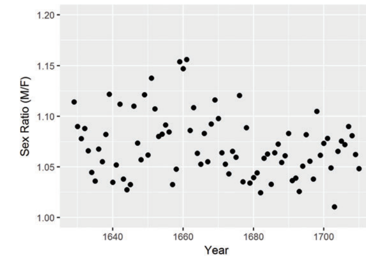
reference line

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

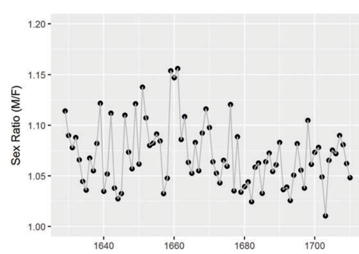
An R script for this example is available at:
https://friendly.github.io/6135/R/arbuthnot-gg.R



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

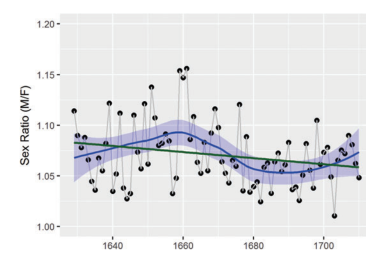


Building a custom graph

```
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:

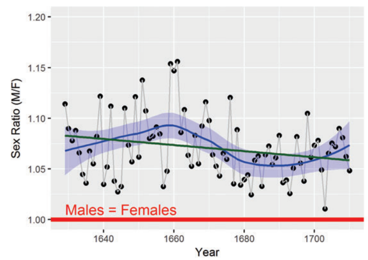
- loess curve
- linear regression line



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)
```

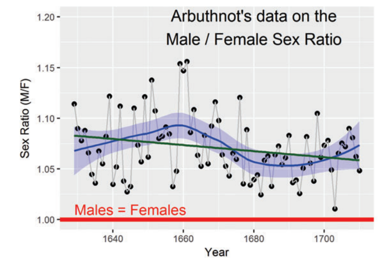
Add horizontal reference line & text label



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)
```

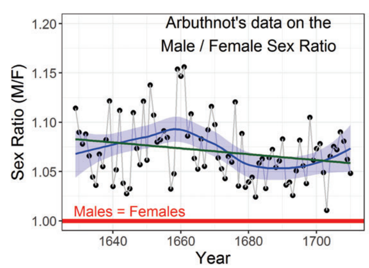
Add figure title



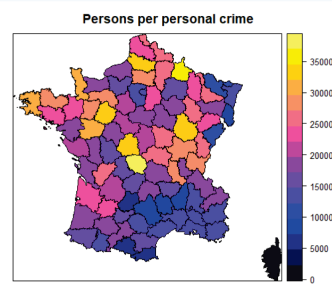
Building a custom graph

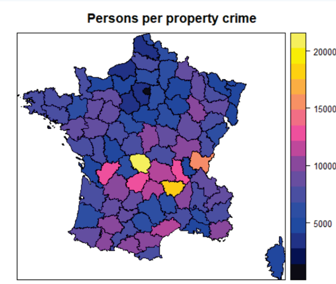
```
arbuth +
  geom_hline(yintercept=1, color="red", size=2) +
  annotate("text", x=1645, y=1.01, label="Males = Females", color="red", size=5) +
  annotate("text", x=1680, y=1.19,
       label="Arbuthnot's data on the\nMale / Female Sex Ratio", size=5.5) +
  theme_bw() + theme(text = element_text(size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

Change the theme and font size



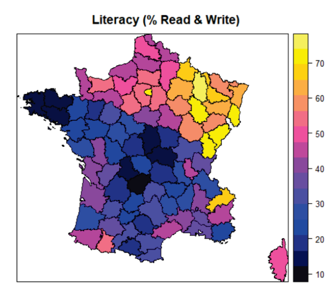
Guerry: Moral statistics of France





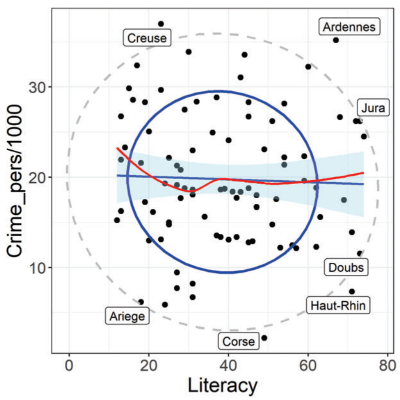
Guerry (1833) made shaded maps of France to determine if crime was related to literacy & other factors

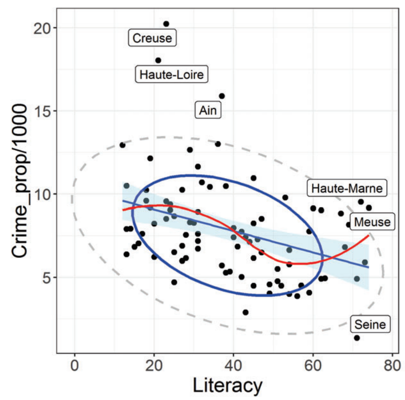
```
library(Guerry)
library(sp)
spplot(gfrance, "Crime_pers")
spplot(gfrance, "Crime_prop")
spplot(gfrance, "Literacy")
```



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!

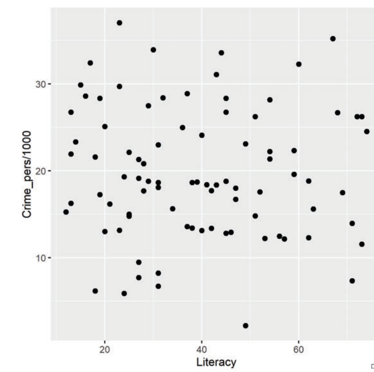




Building Guerry's plots

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

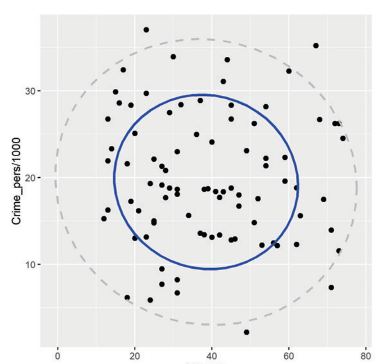
Start with a basic scatterplot



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



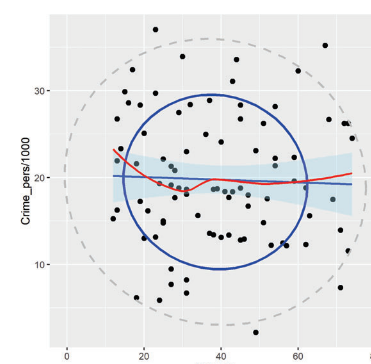
Guerry's plots: Add smooths

```
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()
```

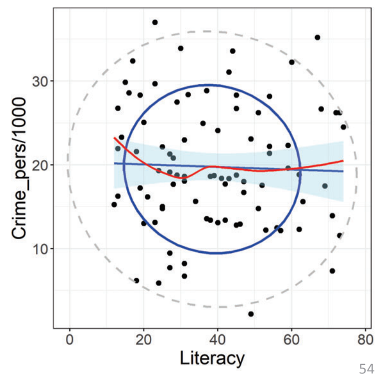


Guerry's plots: Styling

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



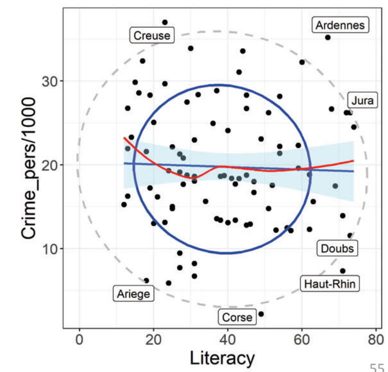
Guerry's plots: Labeling

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,])
```

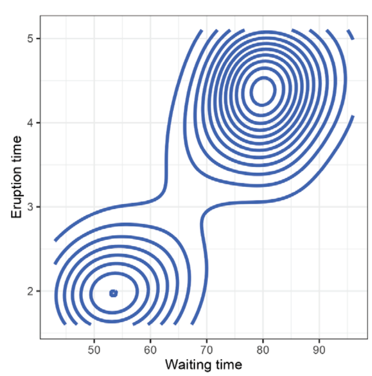


Beyond 2D: Contour plots

Data on eruptions of geysers at Old Faithful (Yellowstone Natl Park)

```
> str(faithful)
'data.frame':        272 obs. of  2 variables:
 $ eruptions: num  3.6 1.8 3.33 2.28 4.53 ...
 $ waiting  : num  79 54 74 62 85 55 88 85 51 85 ...
```

```
p <- ggplot(faithful, aes(waiting, eruptions))

p + geom_density_2d(size=1.2) +
   labs(x = "Waiting time",
        y = "Eruption time")
```



geom_density_2d() calculates relative frequency (density) over a grid of X, Y values

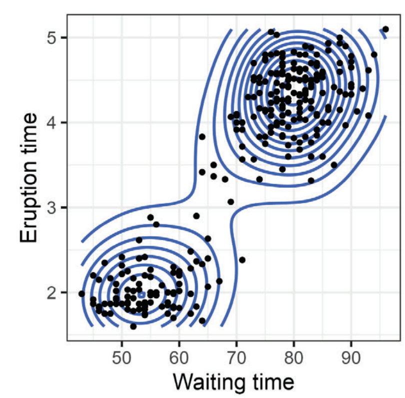
It plots contours of equal density

Contours: add another layer

We also want to see the data: add points

```
p + geom_density_2d(size=1) +
    geom_point() +
    labs(x = "Waiting time",
         y = "Eruption time") +
  theme_bw(base_size = 16)
```

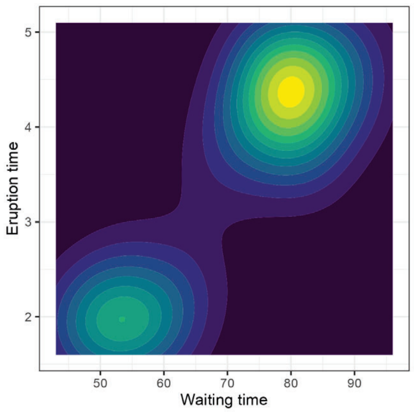
Note use of base_size to control all size elements

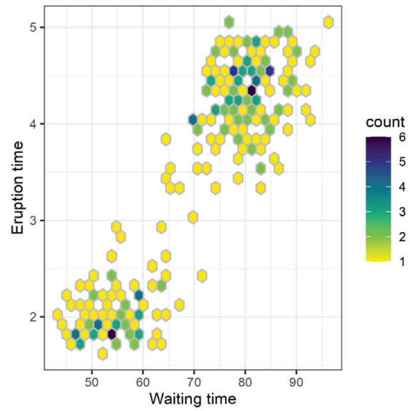


Other forms: Filled contours, hex bins

```
p +
  geom_density_2d_filled(show.legend
                          = FALSE)
```

```
p +
  geom_hex(bins=30, color="gray") +
  scale_fill_viridis_c(direction = -1) +
```

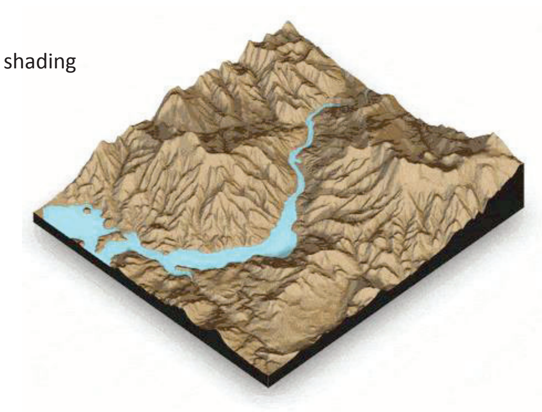




Real 3D: rayshader

The rayshader package produces stunning 3D visualizations of X, Y, Z data
Z (“elevation”) data above an X, Y plane

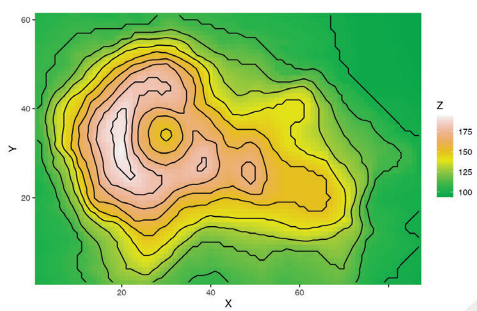
Allows:
- photo-realistic lighting & shading
- animation
- camera motion
- cinematic depth of field



⚠ computationally intensive

rayshader::plot_gg()

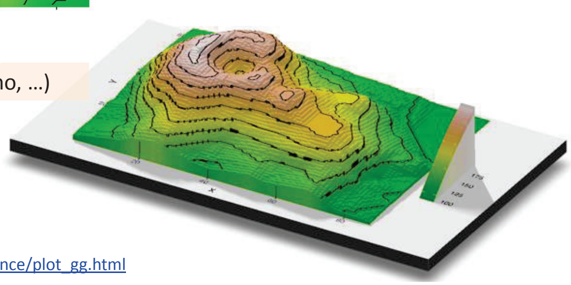
A volcano:



plot_gg(): Plots a ggplot2 object in 3D by mapping the color or fill aesthetic to elevation.

```
ggvolcano <-
  volcano %>%
  ggplot() + geom_contour(...)
```

```
plot_gg(ggvolcano, ...)
```

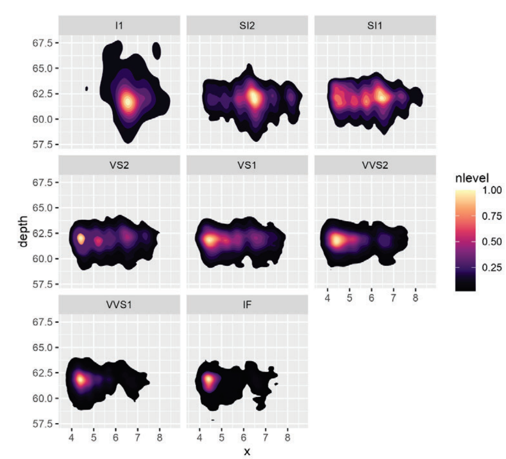


From: https://www.rayshader.com/reference/plot_gg.html

Give your data the full 3D treatment (if you dare)

```
ggdiamonds <-
  ggplot(diamonds, aes(x, depth)) +
  stat_2d_density(...) +
  facet_wrap(clarity ~ .)
```

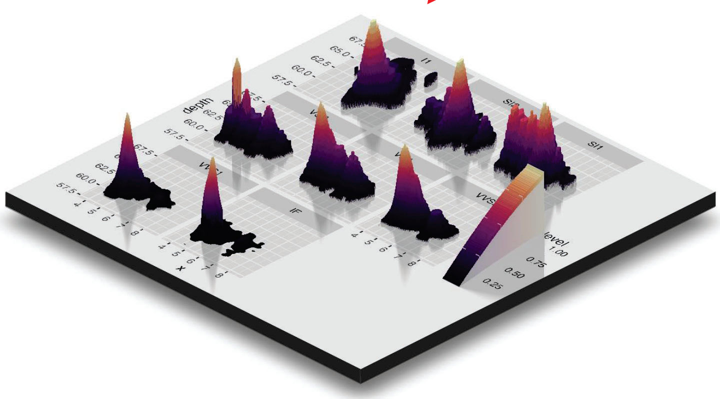
“Please Ma, can I have it in 3D?”



Give your data the full 3D treatment (if you dare)

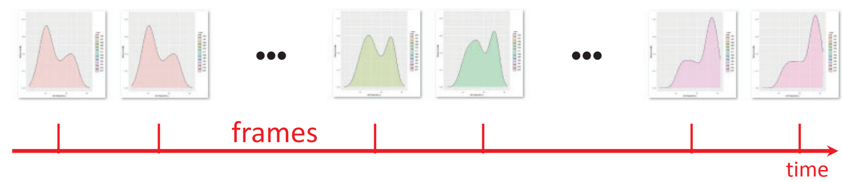
```
ggdiamonds <-
  ggplot(diamonds, aes(x, depth)) +
  stat_2d_density(...) +
  facet_wrap(clarity ~ .)
```

```
plot_gg(ggdiamonds, zoom = 0.55,
        phi = 30, ...)
```



Animation

An animation is a sequence of frames showing a series of related images



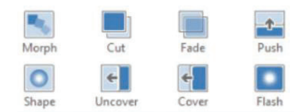
Frames can be based on:

- data subsets (year)
- plot attributes (color, shape)
- plot elements or layers
- ...

Enter / exit types:

- fade
- grow / shrink
- fly / drift

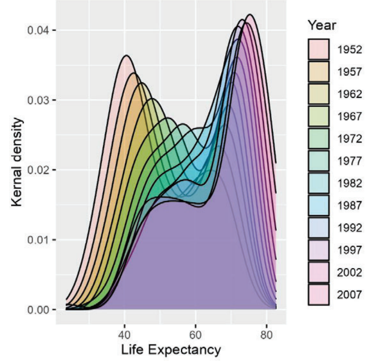
Transition effects:



Animation with gganimate

Draw density curves for life expectancy over years
-- separate curve for each year

```
data(gapminder, package="gapminder")
p <- gapminder %>%
  ggplot(aes(lifeExp, fill = factor(year))) +
  geom_density(alpha=.2) +
  xlab("Life Expectancy") +
  ylab("Kernal density") +
  guides(fill = guide_legend(title = "Year"))
plot(p)
```



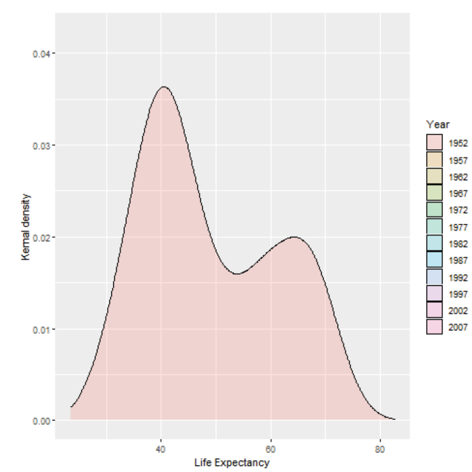
This is pretty, but there is too much going on
-> Animate to show changes over years

Make it an animation

Frames are defined by transition_*() functions

enter_*, exit_*, ease_* functions define the change between transition states

```
p +
  transition_time(year) +
  ease_aes('linear')

anim_save("gap-anim.gif",
          nframes=20)
```



Frames: transition_*()

transition_*()

transition_states()

```
a + transition_states(color, transition_length = 3, state_length = 1)
```

We're cycling between values of color, …

… and spending 3 times as long going to the next cut as we do pausing there.

transition_time()

```
b + transition_time(year, range = c(2002L,2006L))
```

We're cycling through each year of the data…

…from 2002 to 2006 (range is optional; default is the whole time frame). Unlike `transition_states()`, `transition_time()` treats the data as continuous and so the transition length is based on the actual values. Using 2002L instead of 2002 because the underlying data is an integer.

transition_reveal()

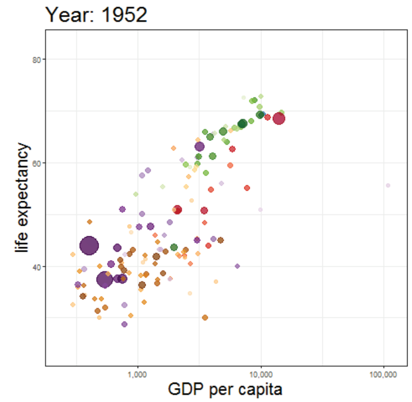
```
c + transition_reveal(date)
```

We're adding each date of the data on top of 'old' data

transition_time()

The classic Rosling moving bubble plot

```
plt <-
ggplot(gapminder,
      aes(x = gdpPercap, y = lifeExp,
          size = pop, colour = country)) +
 geom_point(alpha = 0.7) +
 scale_colour(values=country_colors) +
 scale_size(range = c(2, 15)) +
 scale_x_log10(labels = scales::comma) +
 theme_bw() +
 theme(legend.position = "none",
      plot.title = element_text(size = 25),
      axis.title = element_text(size = 20 ))

plt +
  labs(title = 'Year: {frame_time}') +
  transition_time(year) +
  ease_aes('linear')
```

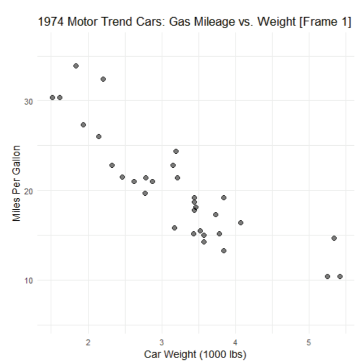


transition_layers()

transition_layers(): Add layers or geoms one at a time

```
ggplot(mtcars, aes(y=mpg, x=wt)) +
 geom_point(size=3, alpha=0.5) +
 labs(x="Car Weight (1000 lbs)",
     y="Miles Per Gallon") +
 ggtitle("1974 Motor Trend Cars: Gas Mileage
          vs. Weight [Frame {frame} ]") +
 theme_minimal(base_size = 14) +
 geom_smooth(method = "lm",  color="red") +
 geom_smooth(method = "loess", color="blue") +

 # do the animation, by layers
 transition_layers() +
 enter_fade() +
 enter_grow()
```

element grows as it enters



Summary

- Understand ggplot components
  - geoms, stats, scales, …
  - think in terms of layers:
    - + geom_point(), + geom_line(), + geom_smooth()
  - themes
    - basic themes, extension packages
    - theme components
- Going beyond 2D
- Animation