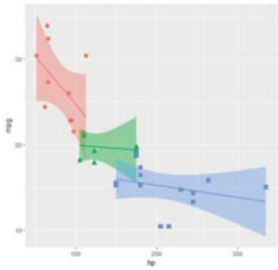
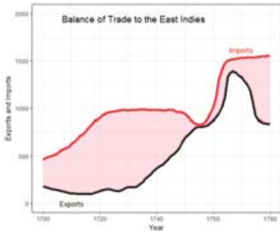
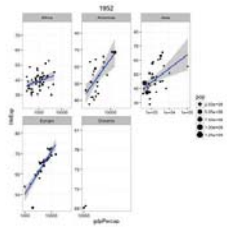






Data Visualization in R

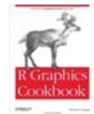
4. ggplot2

Michael Friendly
SCS Short Course
Sep/Oct, 2018

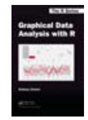
http://www.datavis.ca/courses/RGraphics/

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/



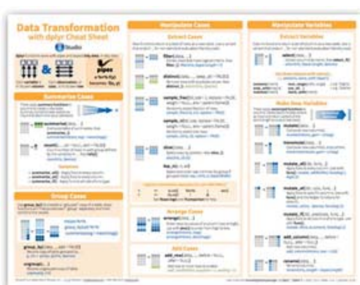
Winston Chang, *R Graphics Cookbook: Practical Recipes for Visualizing Data*
Cookbook format, covering common graphing tasks; the main focus is on ggplot2
R code from book: http://www.cookbook-r.com/Graphs/
Download from: http://ase.tufts.edu/bugs/guide/assets/R%20Graphics%20Cookbook.pdf



Antony Unwin, Graphical Data Analysis with R
R code: http://www.gradaanwr.net/

Resources: Cheat sheets

- Data visualization with ggplot2:
  https://www.rstudio.com/wp-content/uploads/2016/11/ggplot2-cheatsheet-2.1.pdf
- Data transformation with dplyr:
  https://github.com/rstudio/cheatsheets/raw/master/source/pdfs/data-transformation-cheatsheet.pdf



What is ggplot2?

- ggplot2 is Hadley Wickham's R package for producing "elegant graphics for data analysis"
  - It is an implementation of many 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: http://www.ggplot2-exts.org/gallery/

Follow along

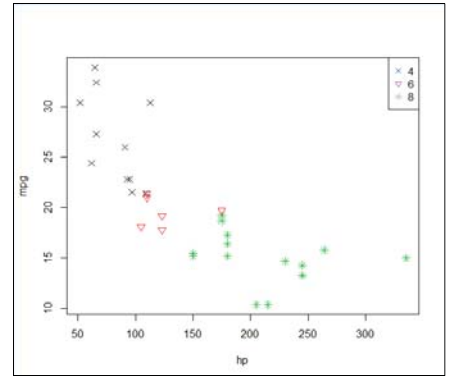
- From the course web page, click on the script gg-cars.R, http://www.datavis.ca/courses/RGraphics/R/gg-cars.R
- Select all (ctrl+A) and copy (ctrl+C) to the clipboard
- In R Studio, open a new R script file (ctrl+shift+N)
- Paste the contents (ctrl+V)
- Run the lines (ctrl+Enter) to along with me

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))
```

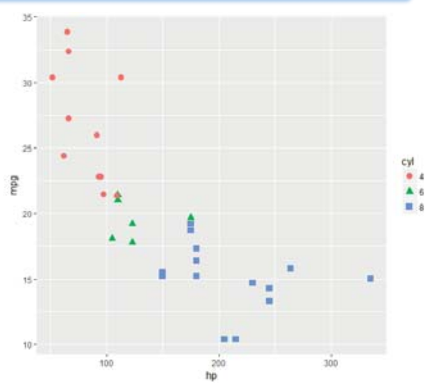
colors and point symbols work differently in plot() and legend()

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

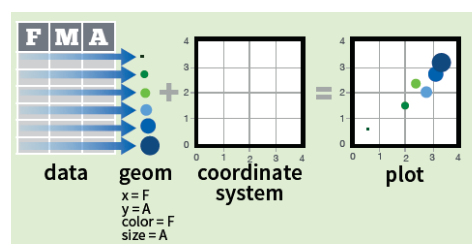
aes() mappings set in the call to ggplot() are passed to geom_point() here




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

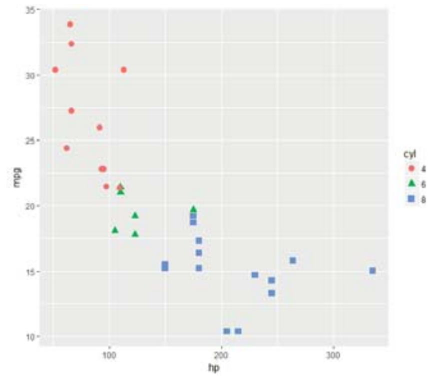

Grammar of Graphics

- 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)

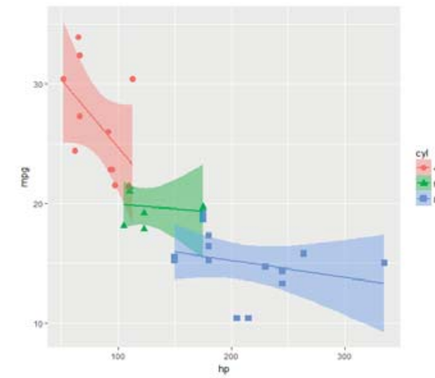
ggplot2: geoms

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

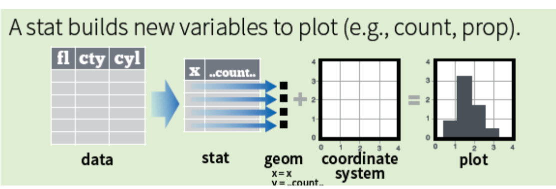
More generally: think of adding new layers to make a plot more useful.




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

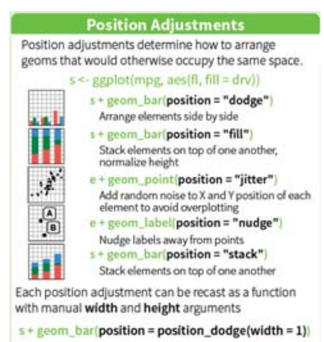

Grammar of Graphics

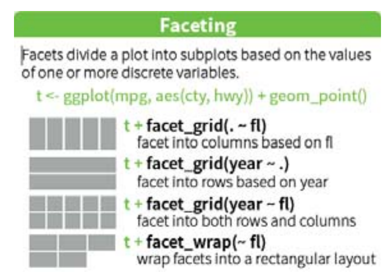
- Other GoG building blocks:
  - statistical transformations ("stat") -- data summaries: mean, sd, binning & counting, …
  - scales: legends, axes to allow reading data from a plot



Grammar of Graphics

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



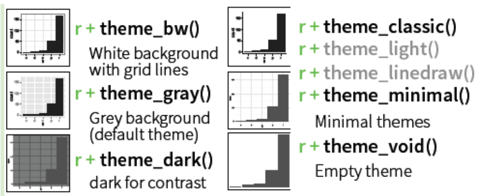


ggplot2: GoG -> graphic language

- 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

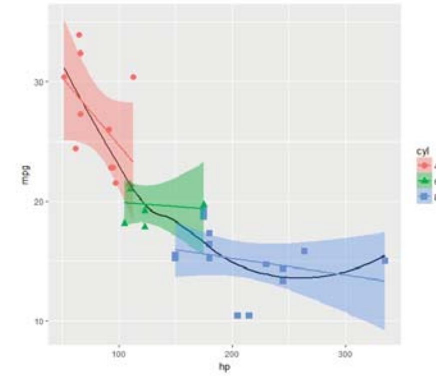


ggplot2: layers & aes()

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
Specifying color= overrides other layers



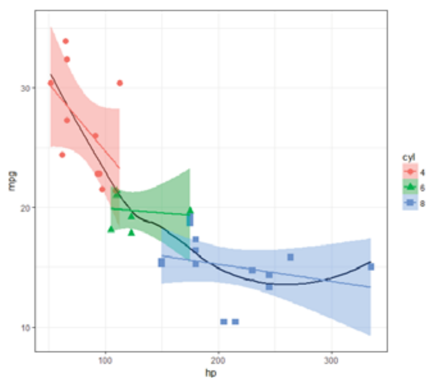

```
ggplot(mtcars, aes(x=hp, y=mpg)) +
  geom_point(size=3, aes(color=cyl, shape=cyl)) +
  geom_smooth(method="lm", aes(color=cyl, fill=cyl)) +
  geom_smooth(method="loess", color="black", se=FALSE)
```


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()
```

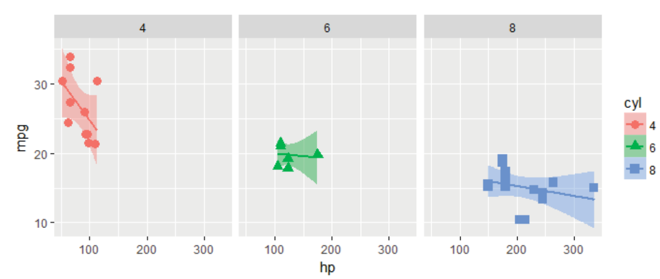

ggplot2: facets

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)
```

Faceting is most useful with other variables, not used in the main plot



labeling points: geom_text()

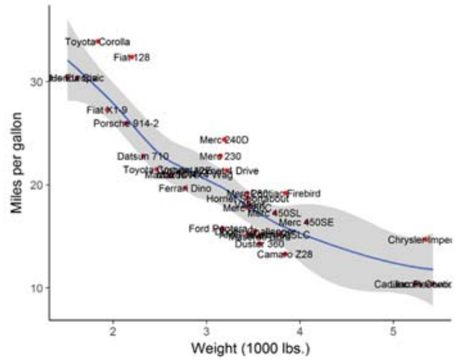
```
plt2 <- ggplot(mtcars, aes(x=wt, y=mpg)) +
   geom_point(color = 'red', size=2) +
   geom_smooth(method="loess") +
   labs(y="Miles per gallon", x="Weight (1000 lbs.)") +
   theme_classic(base_size = 16)

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

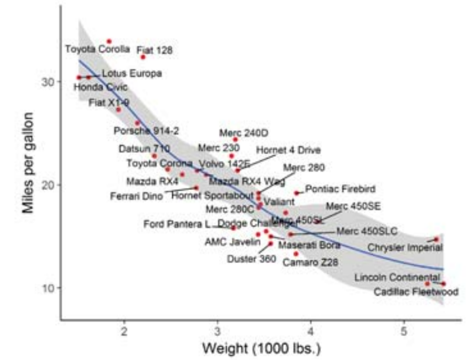


labeling points: geom_text_repel()

```
install.packages("ggrepel")
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

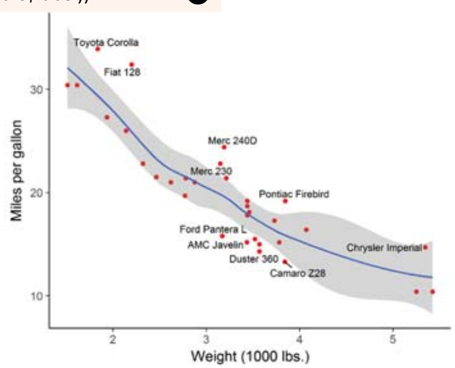
It is easy to label points selectively, using some criterion to assign labels to points

```
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

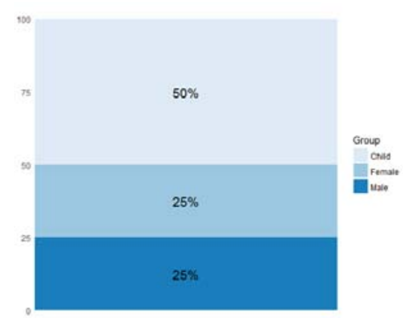


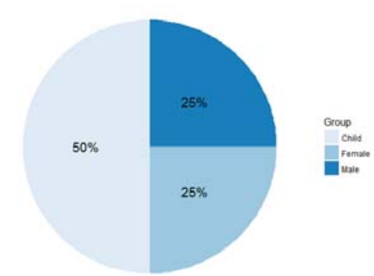
ggplot2: coords

Coordinate systems, coord_*() functions, handle conversion from geometric objects to what you see on a 2D plot.
A pie chart is just a bar chart in polar coordinates!

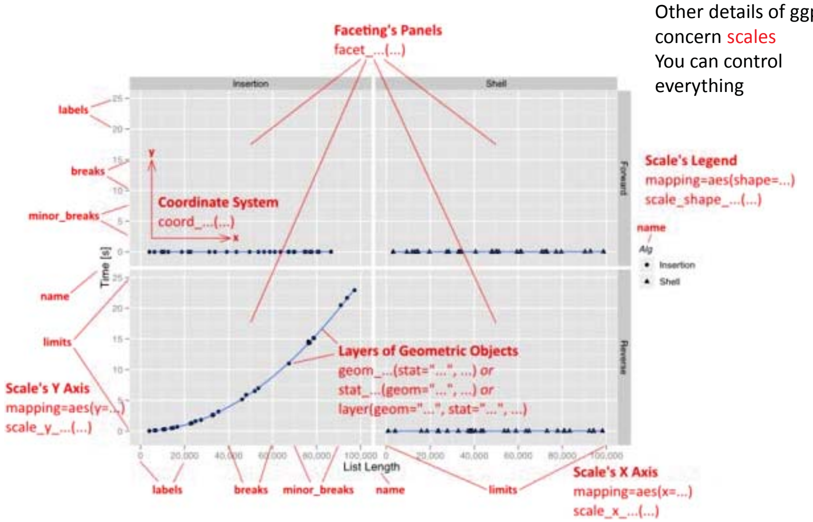
```
p <- ggplot(df, aes(x = "", y = value, fill = group)) +
        geom_bar( stat = "identity")
```

```
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
```

This is what makes layers work with '+'

Normal methods for plot-type objects.
summary() gives some useful info

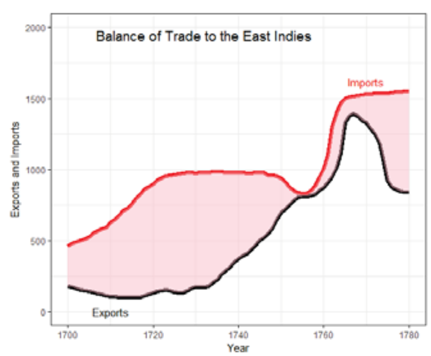
Playfair: Balance of trade charts

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
   ...     ...     ...
```

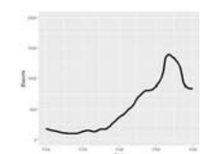
ggplot thinking:
- what are the elements of this graph?
- how can I do them?



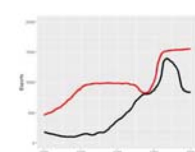
ggplot thinking

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

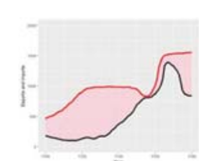


```
c1 <-
ggplot(EastIndiesTrade, aes(x=Year, y=Exports)) +
         ylim(0,2000) +
         geom_line(colour="black", size=2) +
         geom_line(aes(x=Year, y=Imports), colour="red", size=2)
```

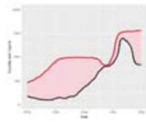


- Fill the area between the curves: geom_ribbon()
- change the Y label

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



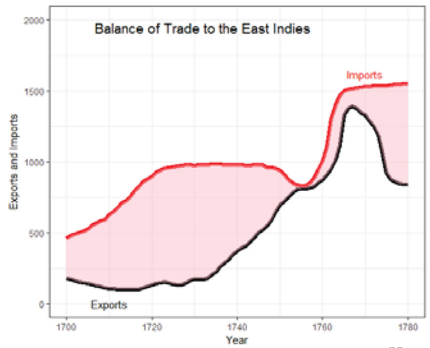
This looks pretty good.
Add some text labels using annotate()



```
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()
```

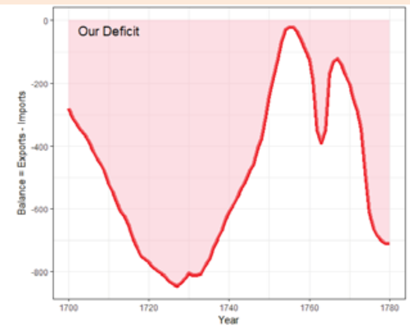


Plot what you want to show

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()
```

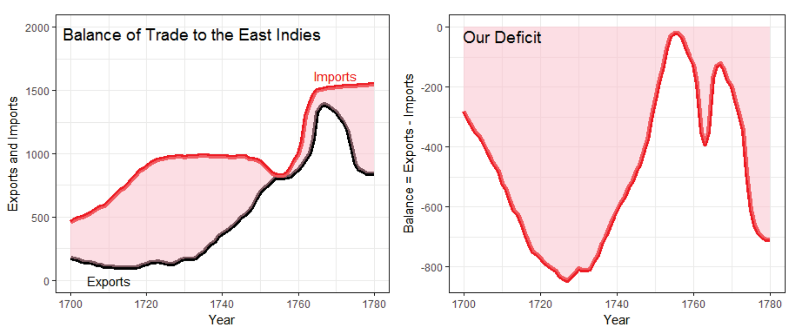
Questions:
- what are the basic plot variables?
- how did I make it shade above the curve?



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")
- If you have a plot object

  ggsave(myplot, file="path/filename.png")
- Specify size:

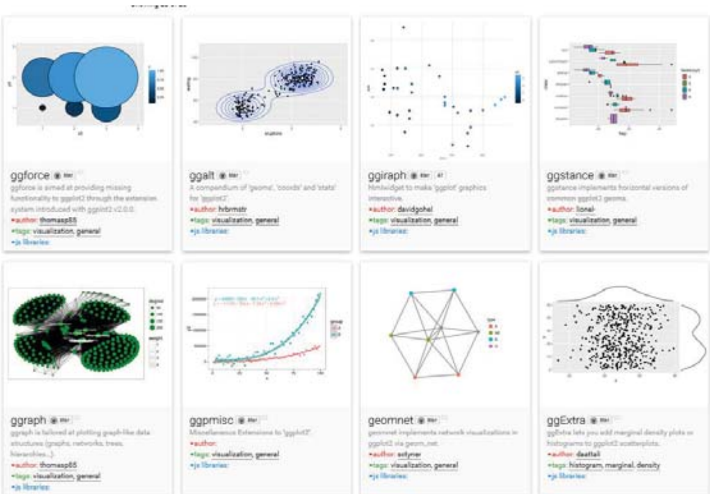
  ggsave(myplot, "path/filename.png", width=6, height=4)
- any plot format (pdf, png, eps, svg, jpg, …)

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

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

ggplot extensions

There are a large number of ggplot extensions. See: http://www.ggplot2-exts.org/



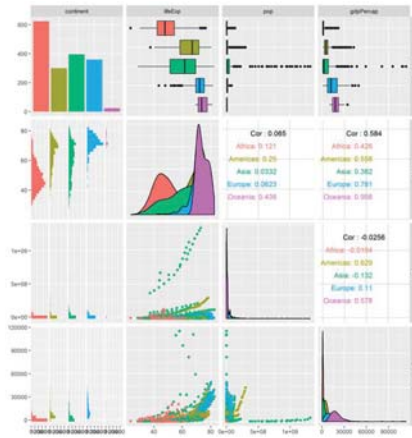
ggplot extensions: GGally

GGally contains a large number of functions that extend ggplot2 to multivariate data

ggpairs() produces generalized scatterplot matrices, with lots of options

```
library(GGally)
library(dplyr)
library(ggplot2)
library(gapminder)

gapminder %>%
   select(-country, -year) %>%
   ggpairs(aes(color=continent))
```

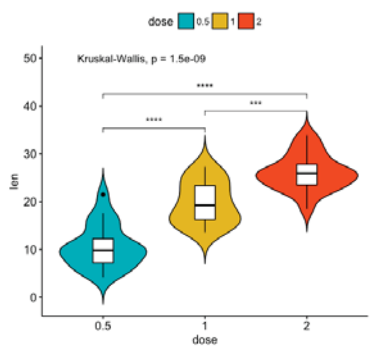


ggpubr

The ggpubr package provides some easy-to-use functions for creating and customizing publication ready plots.

```
ggviolin(df, x = "dose", y = "len", fill = "dose",
        palette = c("#00AFBB", "#E7B800", "#FC4E07"),
        add = "boxplot", add.params = list(fill = "white"))  +
  stat_compare_means(comparisons = my_comparisons, label = "p.signif") +
  stat_compare_means(label.y = 50)
```

see the examples at
http://www.sthda.com/english/rpkgs/ggpubr/

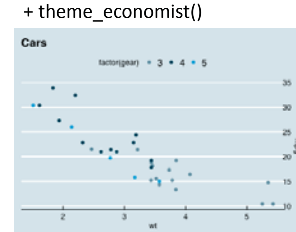


ggthemes

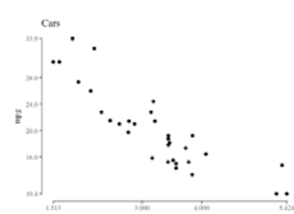
ggthemes provides a large number of extra geoms, scales, and themes for ggplot

```
install.packages('ggthemes', dependencies = TRUE)
```

+ theme_economist()



+ theme_tufte()



+ theme_fivethirtyeight()

Tables in R

- Not a ggplot topic, but it is useful to know that you can also produce beautiful tables in R
- There are many packages for this: See the CRAN Task View on Reproducible Research, https://cran.r-project.org/web/views/ReproducibleResearch.html
  - xtable: Exports tables to LaTeX or HTML, with lots of control
  - stargazer: Well-formatted model summary tables, side-by-side
  - apaStyle: Generate APA Tables for MS Word
- Every time you cut & paste …
  - … God kills a kitten

Tables in R: xtable

Just a few examples, stolen from xtable: vignette("xtableGallery.pdf")

```
fm1 <- aov(tlimth ~ sex + ethnicty + grade + disadvg, data = tli)
xtable(fm1)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
sex	1	75.37	75.37	0.38	0.5417
ethnicty	3	2572.15	857.38	4.27	0.0072
grade	1	36.31	36.31	0.18	0.6717
disadvg	1	59.30	59.30	0.30	0.5882
Residuals	93	18682.87	200.89		

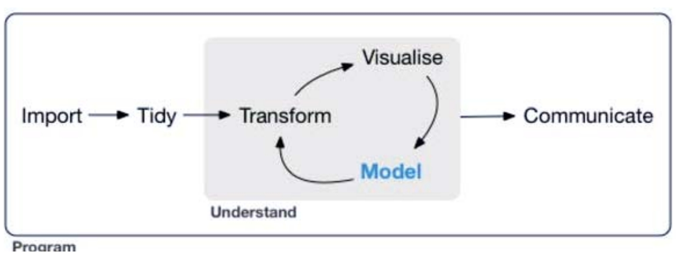
```
fm3 <- glm(disadvg ~ ethnicty*grade, data = tli, family = binomial)
xtable(fm3)
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	3.1888	1.5966	2.00	0.0458
ethnictyHISPANIC	-0.2848	2.4808	-0.11	0.9086
ethnictyOTHER	212.1701	22122.7093	0.01	0.9923
ethnictyWHITE	-8.8150	3.3355	-2.64	0.0082
grade	-0.5308	0.2892	-1.84	0.0665
ethnictyHISPANIC:grade	0.2448	0.4357	0.56	0.5742
ethnictyOTHER:grade	-32.6014	3393.4687	-0.01	0.9923
ethnictyWHITE:grade	1.0171	0.5185	1.96	0.0498

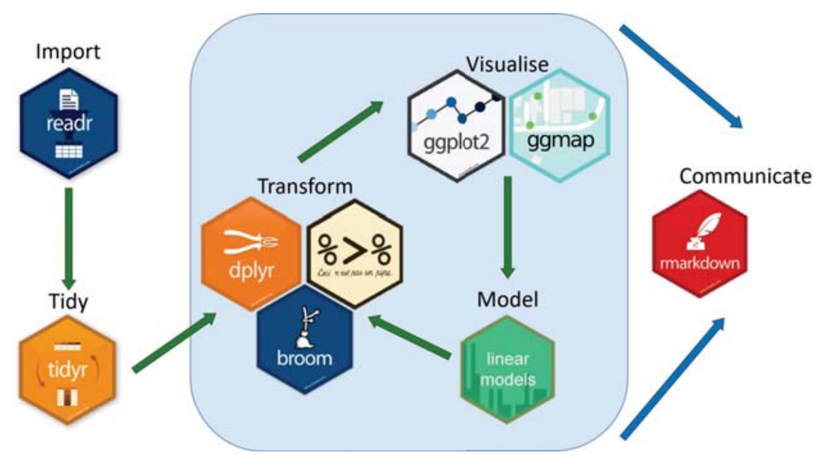
Too many decimals are used here, but you can control all that

A larger view: Data science

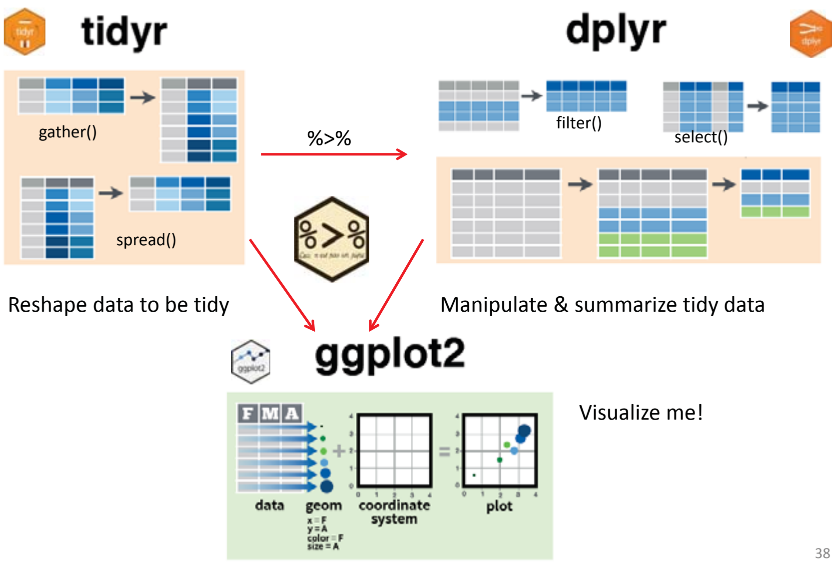
- Data science treats statistics & data visualization as parts of a larger process
  - Data import: text files, data bases, web scraping, …
  - Data cleaning → "tidy data"
  - Model building & visualization
  - Reproducible report writing



The tidyverse of R packages



Tidy tools: overview

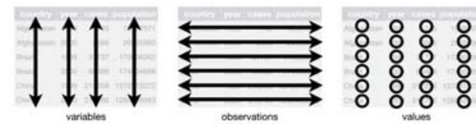


Data wrangling with dplyr & tidyr

What is Tidy Data?

A dataset is said to be tidy if:

- observations are in rows
- variables are in columns
- each value is in its own cell.



A "messy" dataset: Survey of income by religion from Pew Research

- Values of income are in separate columns
- Column headers are values, not variable names
- Cell values are frequencies--- implicit, not explicit

religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k
Agnostic	27	34	60	81	76	137
Atheist	12	27	37	52	35	70
Buddhist	27	21	30	34	33	58
Catholic	418	617	732	670	638	1116

This organization is easy in Excel

But, this makes data analysis and graphing hard

Tidying: reshaping wide to long

We can tidy the data by reshaping from wide to long format using tidyr::gather()

```
> pew <- read.delim(
  file = "http://stat405.had.co.nz/data/pew.txt",
  header = TRUE,
  stringsAsFactors = FALSE, check.names = FALSE)

> (pew1 <- pew[1:4, 1:6])  # small subset

  religion <$10k $10-20k $20-30k $30-40k $40-50k
1 Agnostic    27      34      60      81      76
2  Atheist    12      27      37      52      35
3 Buddhist    27      21      30      34      33
4 Catholic   418     617     732     670     638
```

key value columns

```
>library(tidyr)
> gather(pew1, "income", "frequency", 2:6)
   religion  income frequency
1  Agnostic   <$10k        27
2   Atheist   <$10k        12
3  Buddhist   <$10k        27
4  Catholic   <$10k       418
5  Agnostic $10-20k        34
6   Atheist $10-20k        27
7  Buddhist $10-20k        21
8  Catholic $10-20k       617
9  Agnostic $20-30k        60
10  Atheist $20-30k        37
11 Buddhist $20-30k        30
12 Catholic $20-30k       732
13 Agnostic $30-40k        81
14  Atheist $30-40k        52
15 Buddhist $30-40k        34
16 Catholic $30-40k       670
…  …        …              …
```

NB: income is a character variable; we might want to create an ordered factor or numeric version

Another solution, using reshape2::melt()

```
> library(reshape2)
> pew_tidy <- melt(
    data = pew1,
    id = "religion",
    variable.name = "income",
    value.name = "frequency"
)
```


Using pipes: %>%

- R is a functional language
  - This means that f(x) returns a value, as in y <- f(x)
  - That value can be passed to another function: g(f(x))
  - And so on: h(g(f(x)))

```
> x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142)
> exp(diff(log(x)))
[1] 3.29 1.75 1.58 0.52 0.28
```

  - This gets messy and hard to read, unless you break it down step by step

```
> # Compute the logarithm of `x`, calculate lagged differences,
> # return the exponential function of the result
> log(x)
[1] -2.216 -1.024 -0.462 -0.004 -0.664 -1.952
> diff(log(x))
[1]  1.19  0.56  0.46 -0.66 -1.29
> exp(diff(log(x)))
[1] 3.29 1.75 1.58 0.52 0.28
```

Using pipes: %>%

- Pipes (%>%) change the syntax to make this easier

```
> # use pipes
> x %>% log() %>% diff() %>% exp()
[1] 3.29 1.75 1.58 0.52 0.28
```

- Basic rules
  - `x %>% f()` passes object on left hand side as **first** argument (or . argument) of function on right hand side
    - `x %>% f()` is the same as `f(x)`
    - `x %>% f(y)` is the same as `f(x, y)`
    - `y %>% f(x, ., z)` is the same as `f(x, y, z)`
  - `x %<>% f()` does the same, but assigns the result to x
    - Shortcut for `x <- x %>% f()`

dplyr: Subset observations (rows)

dplyr implements a variety of verbs to select a subset of observations from a dataset

dplyr::**filter(iris, Sepal.Length > 7)**
Extract rows that meet logical criteria.

dplyr::**distinct(iris)**
Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**
Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**
Randomly select n rows.

dplyr::**slice(iris, 10:15)**
Select rows by position.

dplyr::**top_n(storms, 2, date)**
Select and order top n entries (by group if grouped data).

In a pipe expression, omit the dataset name

```
iris %>% filter(Sepal.Length >7)
iris %>% filter(Species==“setosa”)

iris %>% sample_n(10)
iris %>% slice(1:50)   # setosa
```


dplyr: Subset variables (columns)

dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**
Select columns by name or helper function.

Many helper functions in dplyr allow selection by a function of variable names:

select(iris, **contains(".")**)
Select columns whose name contains a character string.

select(iris, **ends_with("Length")**)
Select columns whose name ends with a character string.

select(iris, **everything()**)
Select every column.

select(iris, **matches(".t.")**)
Select columns whose name matches a regular expression.

select(iris, **num_range("x", 1:5)**)
Select columns named x1, x2, x3, x4, x5.

select(iris, **one_of(c("Species", "Genus"))**)
Select columns whose names are in a group of names.

select(iris, **starts_with("Sepal")**)
Select columns whose name starts with a character string.

select(iris, Sepal.Length**:**Petal.Width)
Select all columns between Sepal.Length and Petal.Width (inclusive).

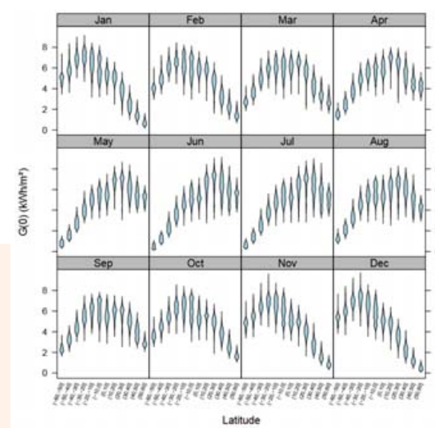
select(iris, **-**Species)
Select all columns except Species.

Faceting & tidy data

Here is a complex graph, showing distributions of solar radiation from NASA, by months of the year and latitude

This is complicated, because the data structure is **untidy**--- months were in separate variables (wide format)

```
> str(nasa)
'data.frame':   64800 obs. of  15 variables:
 $ Lat: int  -90 -90 -90 -90 -90 -90 -90 -90 -90 -90 ...
 $ Lon: int  -180 -179 -178 -177 -176 -175 -174 -173 -172 -171 ...
 $ Jan: num  9.63 9.63 9.63 9.63 9.63 9.63 9.63 9.63 9.63 9.63 ...
 $ Feb: num  5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 ...
 $ Mar: num  0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 ...
 $ Apr: num  0 0 0 0 0 0 0 0 0 0 ...
 $ May: num  0 0 0 0 0 0 0 0 0 0 ...
 $ Jun: num  0 0 0 0 0 0 0 0 0 0 ...
 $ Jul: num  0 0 0 0 0 0 0 0 0 0 ...
 $ Aug: num  0 0 0 0 0 0 0 0 0 0 ...
 $ Sep: num  0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
 $ Oct: num  3.24 3.24 3.24 3.24 3.24 3.24 3.24 3.24 3.24 3.24 ...
 $ Nov: num  8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 ...
 $ Dec: num  11 11 11 11 11 ...
 $ Ann: num  3.19 3.19 3.19 3.19 3.19 3.19 3.19 3.19 3.19 3.19 ...
```



Each distribution is shown as a **violin plot**, a mirrored density plot

tidying the data

In wide format, using lattice, I had to construct a plot formula to plot those columns

```
> x <- paste(names(nasa)[3:14], collapse='+')
> (formula <- as.formula(paste(x, '~cut(Lat, pretty(Lat, 20))', sep='')))
Jan + Feb + Mar + Apr + May + Jun + Jul + Aug + Sep + Oct + Nov +
    Dec ~ cut(Lat, pretty(Lat, 20))
```

Ugh!

It is much easier to reshape the data to long format, so solar is all in one column

```
library(tidyr)
library(dplyr)
library(ggplot2)

nasa_long <- nasa %>%
   select(-Ann) %>%
   gather(month, solar, Jan:Dec, factor_key=TRUE) %>%
   filter( abs(Lat) < 60 ) %>%
   mutate( Lat_f = cut(Lat, pretty(Lat, 12)))
```

%>% "pipes" data to the next stage

select() extracts or drops columns

gather() collapses columns into key-value pairs

filter() subsets observations

mutate() creates new variables

tidying the data

```
> str(nasa_long)
'data.frame':   514080 obs. of  5 variables:
 $ Lat  : int  -59 -59 -59 -59 -59 -59 -59 -59 -59 -59 ...
 $ Lon  : int  -180 -179 -178 -177 -176 -175 -174 -173 -172 -171 ...
 $ month: Factor w/ 12 levels "Jan","Feb","Mar",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ solar: num  5.19 5.19 5.25 5.25 5.17 5.17 5.15 5.15 5.15 5.15 ...
 $ Lat_f: Factor w/ 12 levels "(-60,-50]","(-50,-40]",..: 1 1 1 1 1 1 1 1 1 1 ...

> head(nasa_long)
  Lat  Lon month solar     Lat_f
1 -59 -180   Jan  5.19 (-60,-50]
2 -59 -179   Jan  5.19 (-60,-50]
3 -59 -178   Jan  5.25 (-60,-50]
4 -59 -177   Jan  5.25 (-60,-50]
5 -59 -176   Jan  5.17 (-60,-50]
6 -59 -175   Jan  5.17 (-60,-50]
```

solar is now the single response variable

For ease of plotting, I created a factor version of Lat with 12 levels

The data are now in a form where I can plot solar against Lat or Lat_f and facet by month

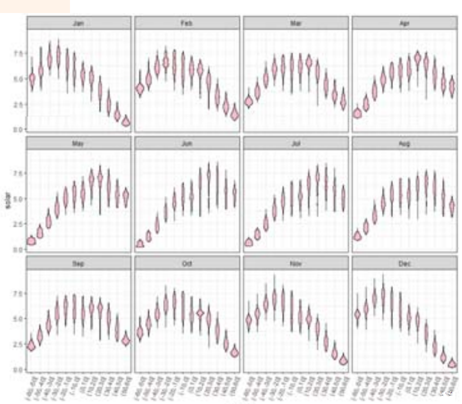
plotting the tidy data

Using geom_violin() shows the shapes of the distributions for levels of Lat_f

```
ggplot(nasa_long, aes(x=Lat_f, y=solar)) +
   geom_violin(fill="pink") +
   facet_wrap(~ month) +
   theme_bw() +
   theme(axis.text.x =
          element_text(angle = 70,
                       hjust = 1))
```

facet_wrap(~month) does the right thing

I had to adjust the x-axis labels for Lat_f to avoid overplotting

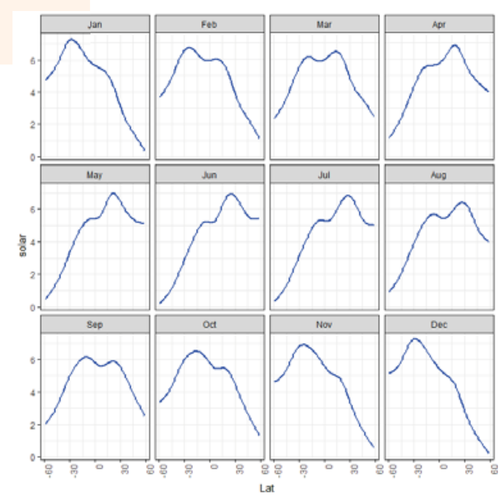


plotting the tidy data: smoothing

```
ggplot(nasa_long, aes(x=Lat, y=solar)) +
   geom_smooth(color="blue" ) +
   facet_wrap(~ month) +
   theme_bw()
```

Here I treat Lat as quantitative geom_smooth() uses method = "gam" here because of large *n*

The variation in the smoothed trends over the year suggest quite lawful behavior



build a model

What we saw in the plot suggests a generalized additive model, with a smooth, s(Lat)

```
library(mgcv)
nasa.gam <- gam(solar ~ Lon + month + s(Lat), data=nasa_long)
summary(nasa.gam)
```

```
Family: gaussian
Link function: identity

Formula:
solar ~ Lon + month + s(Lat)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.691e+00  6.833e-03 686.409  < 2e-16 ***
Lon         -1.713e-04  1.898e-05  -9.022  < 2e-16 ***
monthFeb     1.195e-01  9.664e-03  12.364  < 2e-16 ***
  ...        ...
monthDec    -8.046e-02  9.664e-03  -8.326  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
       edf Ref.df     F p-value
s(Lat) 8.997      9 37285  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.398   Deviance explained = 39.8%
GCV = 2.0006  Scale est. = 2.0005    n = 514080
```

The violin plots suggest that variance is not constant. I'm ignoring this here by using the default gaussian model. (Good first start)

Model terms:

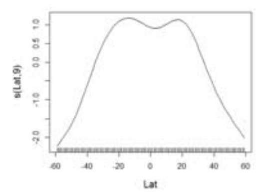
- Lon wasn't included before
- month is a factor, for the plots
- s(Lat) fits a smoothed term in latitude, averaged over other factors

There are other model choices, but it is useful to visualize what we have done so far

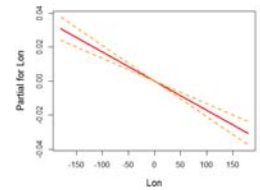
visualize the model

Effect plots show the fitted relationship between the response and model terms, averaged over other predictors.
The mgcv package has its own versions of these.

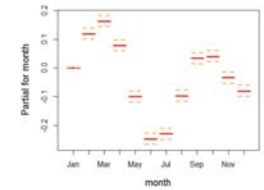
```
plot(nasa.gam, cex.lab=1.25)
termplot(nasa.gam, terms="month", se=TRUE, lwd.term=3, lwd.se=2, cex.lab=1.25)
termplot(nasa.gam, terms="Lon", se=TRUE, lwd.term=3, lwd.se=2, cex.lab=1.25)
```



why the dip at the equator?



effect of longitude is very small, but maybe interpretable



month should be modeled as a time variable

Summary

- ggplot2 provides a new way of thinking about graphs
  - aes() – mapping data variables to visual properties
  - geom_() – drawing geometric objects (points, lines, …)
  - coord_() – transform coordinate systems
  - layers – add stuff to an existing plot with '+'
  - themes – change the entire look of a graph
- tidyr & dplyr provide a new way of thinking about data analysis
- R Studio tools provide a way to organize your work, do analysis, and publish --- reproducible!