

# An introduction to R Graphics

## 4. ggplot2



Michael Friendly  
SCS Short Course  
March, 2017



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

# Resources: Books

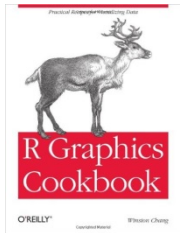


Hadley Wickham, *ggplot2: Elegant graphics for data analysis*, 2nd Ed.

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/>

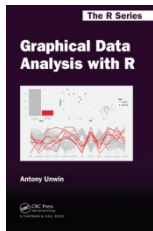


Winston Chang, *R Graphics Cookbook: Practical Recipes for Visualizing Data*

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/>

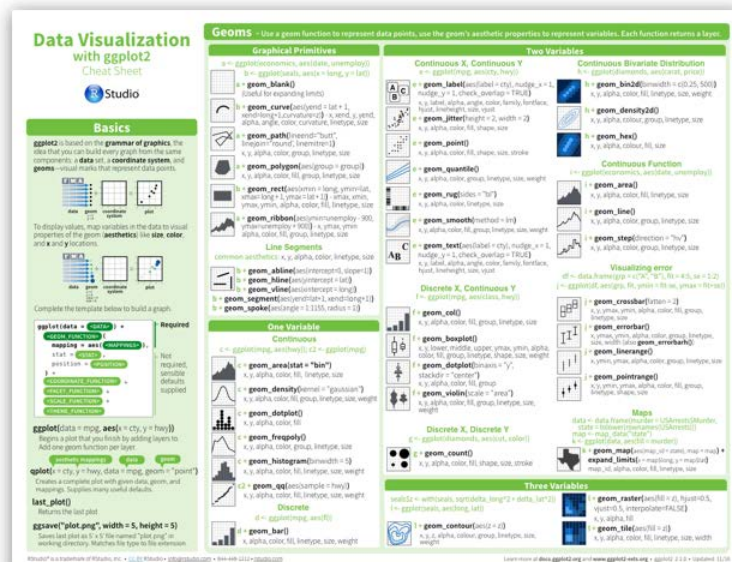
# Resources: Cheat sheets

- 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>



**Data Visualization with ggplot2**  
Cheatsheet  
RStudio

**Basics**  
ggplot() is based on the **grammar of graphics**, the idea that you can build every graphic from three core components: **data** (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, area, and opacity).

Complete the template below to build a graph.

```
ggplot(data = dataset , aes(x = variable , y = variable )) +  
  geom_function(aesthetics)
```

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

**Graphical Primitives**

- `geom_blank()` `geom_point()`
- `geom_curve()` `geom_polygon()`
- `geom_rect()` `geom_ridge()`
- `geom_segment()` `geom_text()`
- `geom_vline()` `geom_xline()`

**Two Variables**

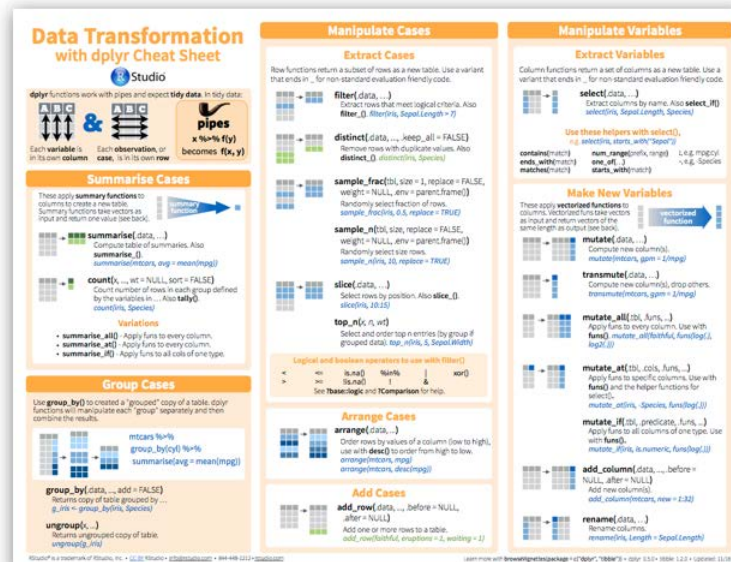
- Continuous X, Continuous Y**  
`geom_line()`, `geom_smooth()`, `geom_point()`, `geom_jitter()`
- Continuous Bivariate Distribution**  
`geom_bin2d()`, `geom_density2d()`
- Continuous Function**  
`geom_line()`, `geom_smooth()`
- Discrete X, Continuous Y**  
`geom_bar()`, `geom_boxplot()`, `geom_histogram()`, `geom_jitter()`
- Discrete X, Discrete Y**  
`geom_tile()`, `geom_count()`

**One Variable**

- Continuous**  
`geom_area()`, `geom_density()`, `geom_freqpoly()`, `geom_histogram()`, `geom_qq()`
- Discrete**  
`geom_bar()`

**Three Variables**

- `geom_raster()`, `geom_tile()`, `geom_violin()`



**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
- Each variable is a vector of values (e.g., `1:5`)

**Summarise Cases**

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

- `summarise()` `summarize()`
- `count()` `tbl_count()`

**Group Cases**

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

- `group_by()` `ungroup()`

**Manipulate Cases**

**Extract Cases**

- `filter()` `filter_at()` `filter_if()`
- `distinct()` `distinct_at()` `distinct_if()`
- `sample_frac()` `sample_n()` `slice()` `slice_sample()` `top_n()`

**Manipulate Variables**

**Extract Variables**

- `select()` `select_at()` `select_if()` `rename()` `rename_at()` `rename_if()`

**Make New Variables**

These apply **vectorized** functions to columns, vector objects, or base vectors as input and return vectors of the same length as the output base table.

- `mutate()` `transmute()` `mutate_at()` `mutate_if()` `mutate_each()` `mutate_at_each()` `mutate_if_each()` `mutate_if_all()` `mutate_if_any()`

# What is ggplot2?

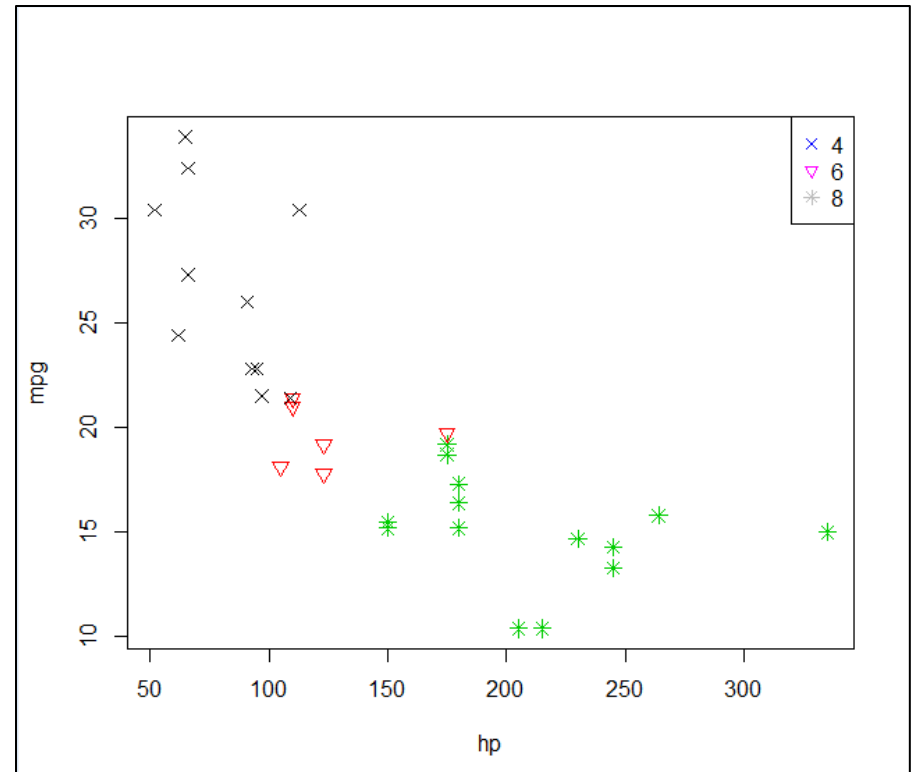
- ggplot2 is Hadley Wickham's R package for producing "elegant graphics for data analysis"
  - It is an implementation of many of the ideas for graphics introduced in Lee Wilkinson's *Grammar of Graphics*
  - These ideas and the syntax of ggplot2 help to think of graphs in a new and more general way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

# 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

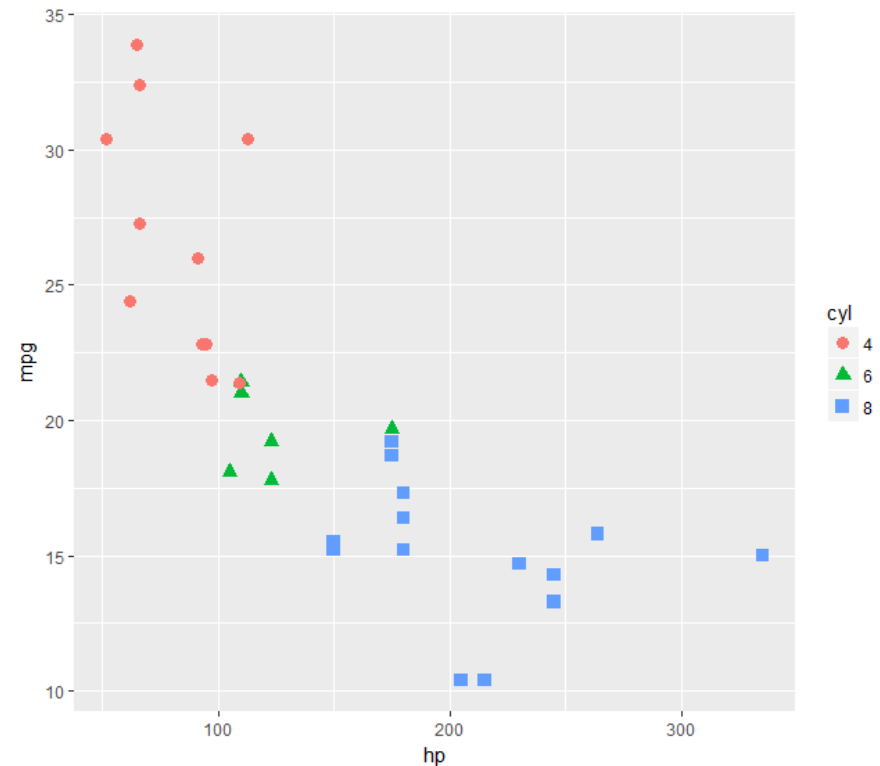
In ggplot2, just map the data variables to aesthetic attributes

`aes(x, y, shape, color, size, ...)`

`ggplot()` takes care of the rest

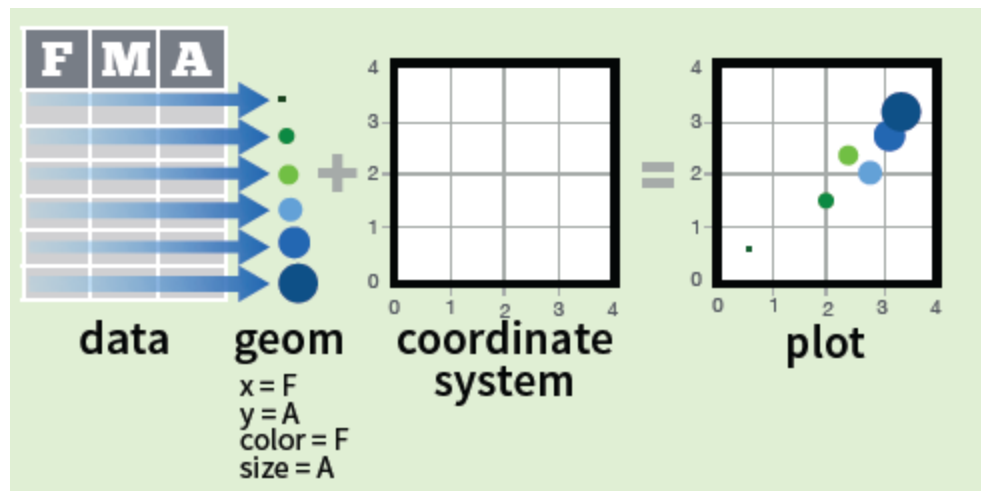
`aes()` mappings set in the call to `ggplot()` are passed to `geom_point()` here

```
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
  - **aes**thetic mapping of variables into visual properties: size, color, x, y
  - **geom**etric objects (“geom”): points, lines, areas, arrows, ...
  - **coord**inate system (“coord”): Cartesian, log, polar, map,

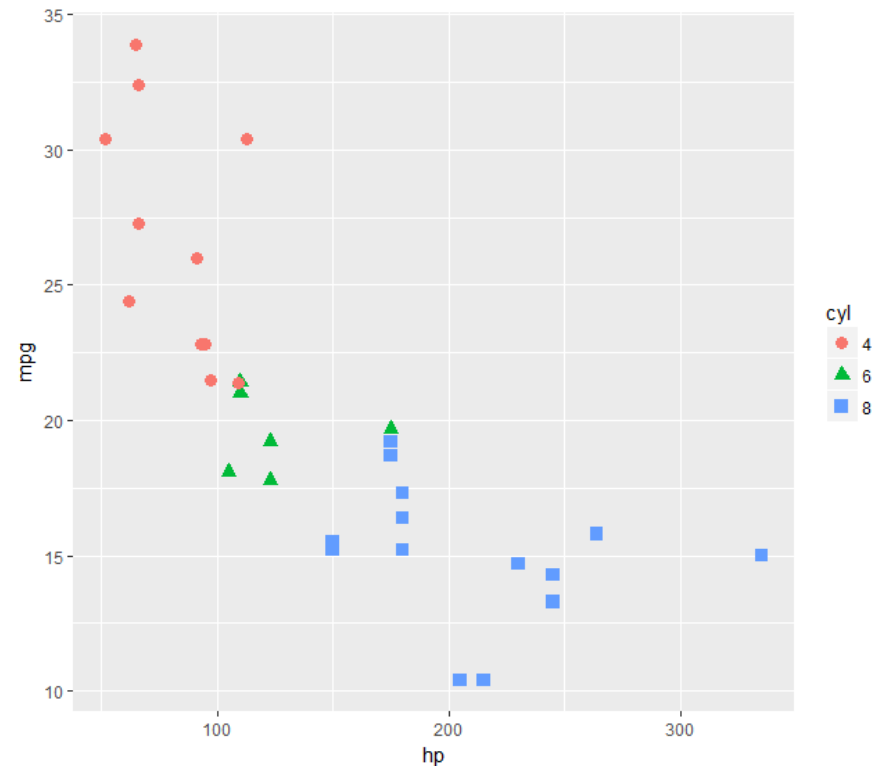


# ggplot2: data + geom -> graph

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

In this call,

- `data=mtcars`: data frame
- `aes(x=hp, y=mpg)`: plot variables
- `aes(color, shape)`: attributes
- `geom_point()`: what to plot
- the coordinate system is taken to be the standard Cartesian (x,y)



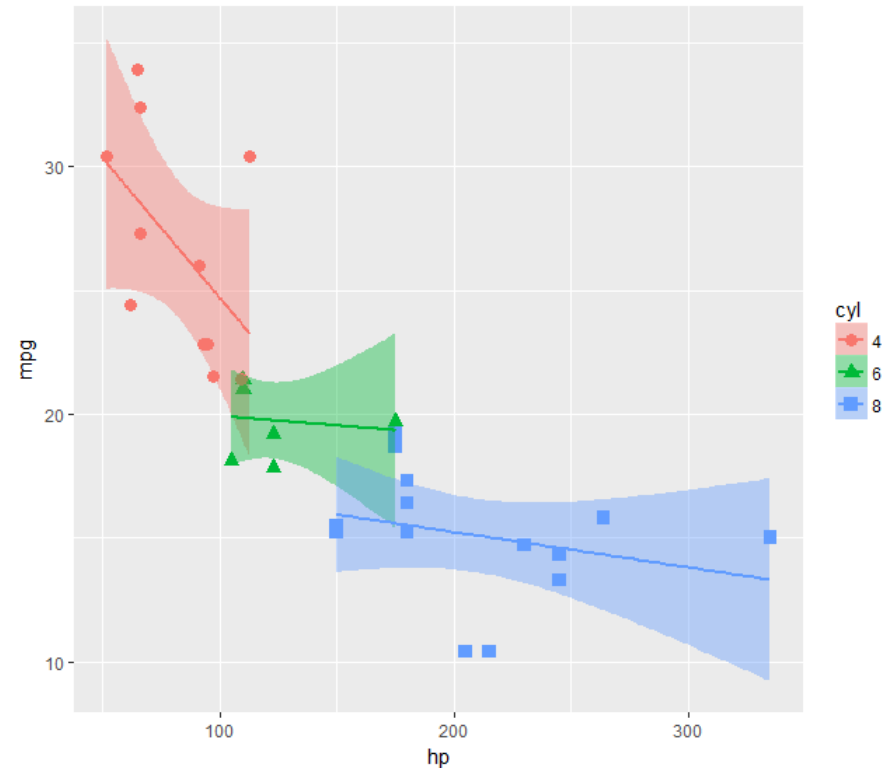


# ggplot2: geoms

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`

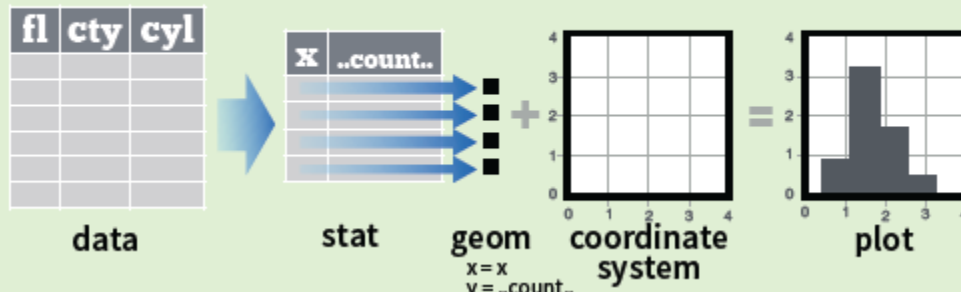


```
ggplot(mtcars, aes(x=hp, y=mpg, color=cyl, shape=cyl)) +  
  geom_point(size=3) +  
  geom_smooth(method="lm", aes(fill=cyl))
```

# Grammar of Graphics

- Other GoG building blocks:
  - **stat**istical transformations (“stat”) -- data summaries: mean, sd, binning & counting, ...
  - **scale**s: 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)
```

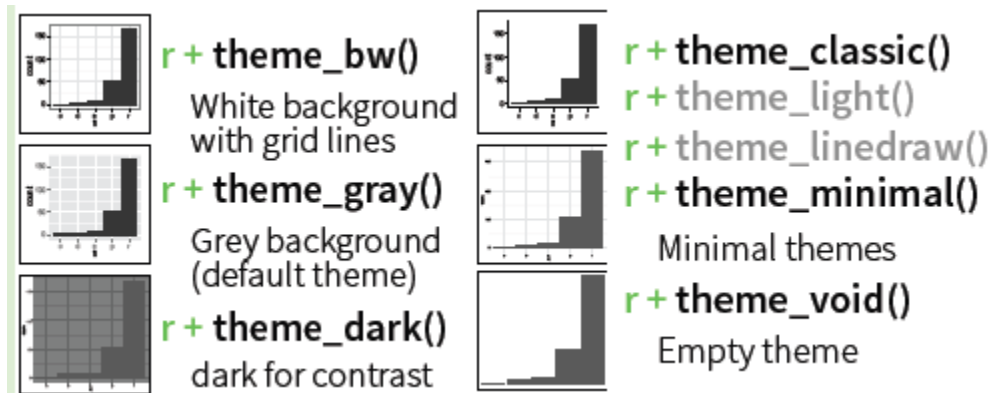
wrap facets into a rectangular layout

# 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") +
```

- **theme**: change graphic elements consistently

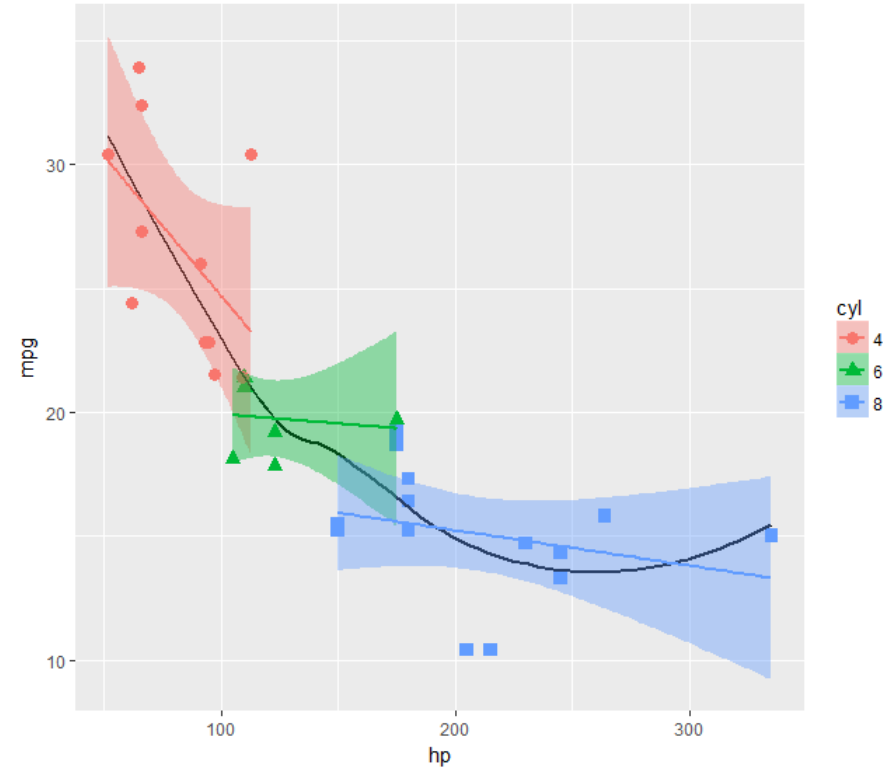


# ggplot2: layers & aes()

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



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

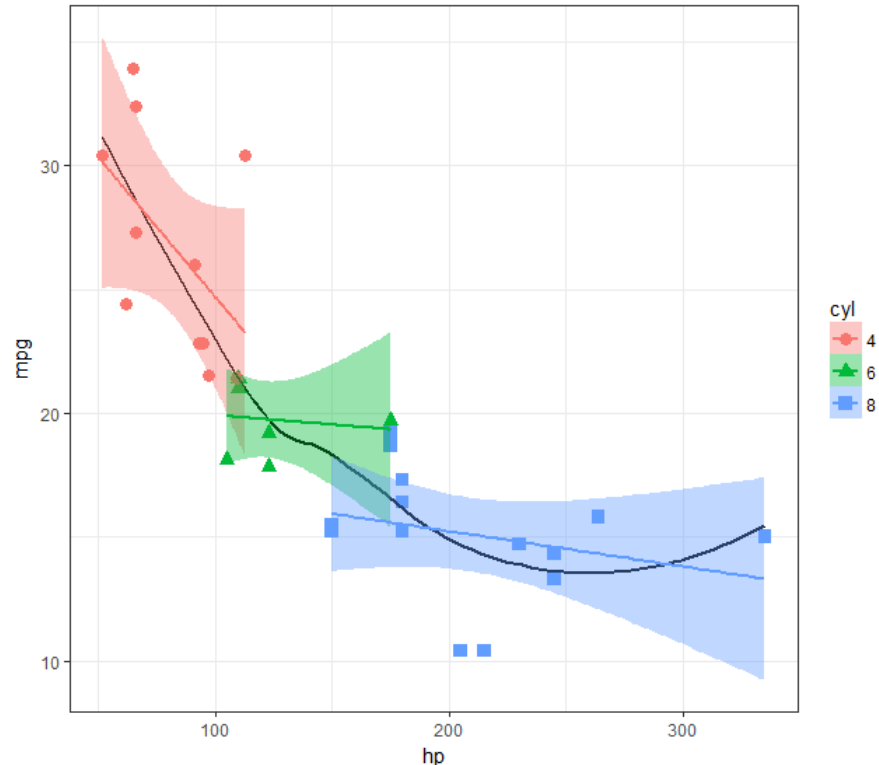
# ggplot2: themes

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()
```





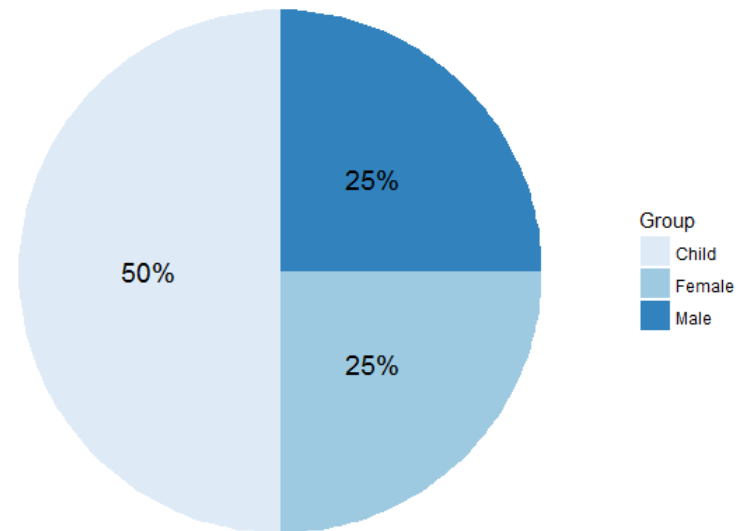
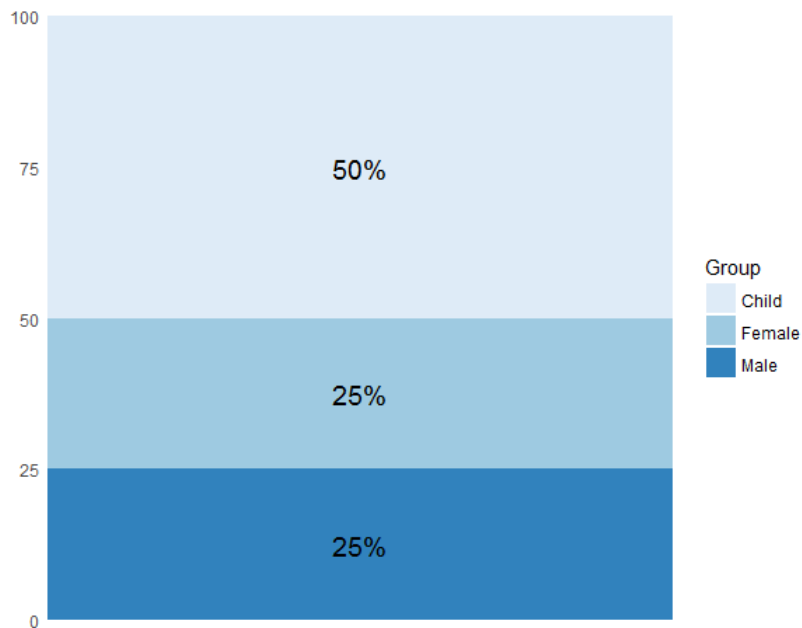
# ggplot2: coords

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!

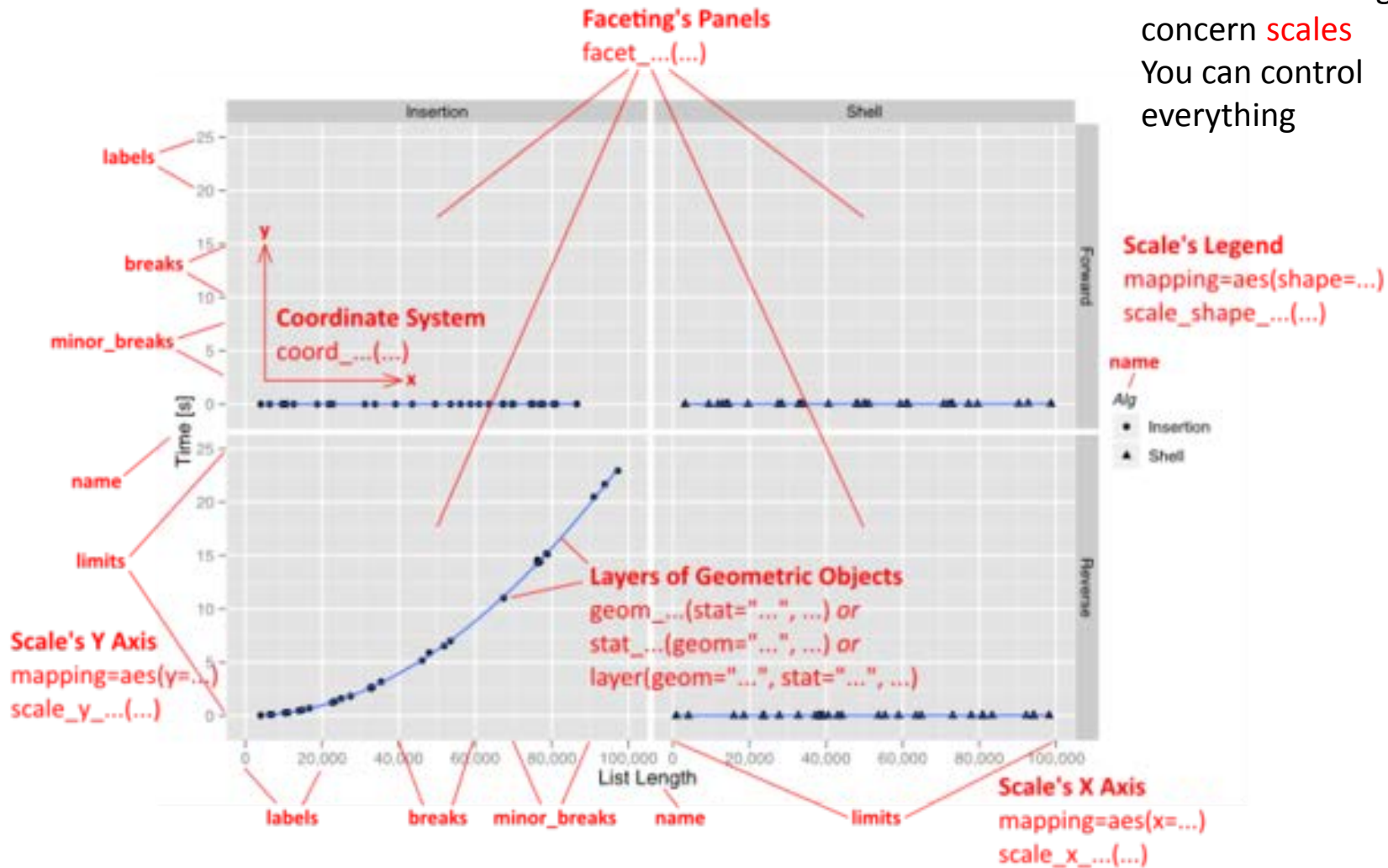
```
p <- ggplot(df, aes(x = "", y = value, fill = group)) +  
  geom_bar(stat = "identity")
```

```
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
```

# Playfair: Balance of trade charts

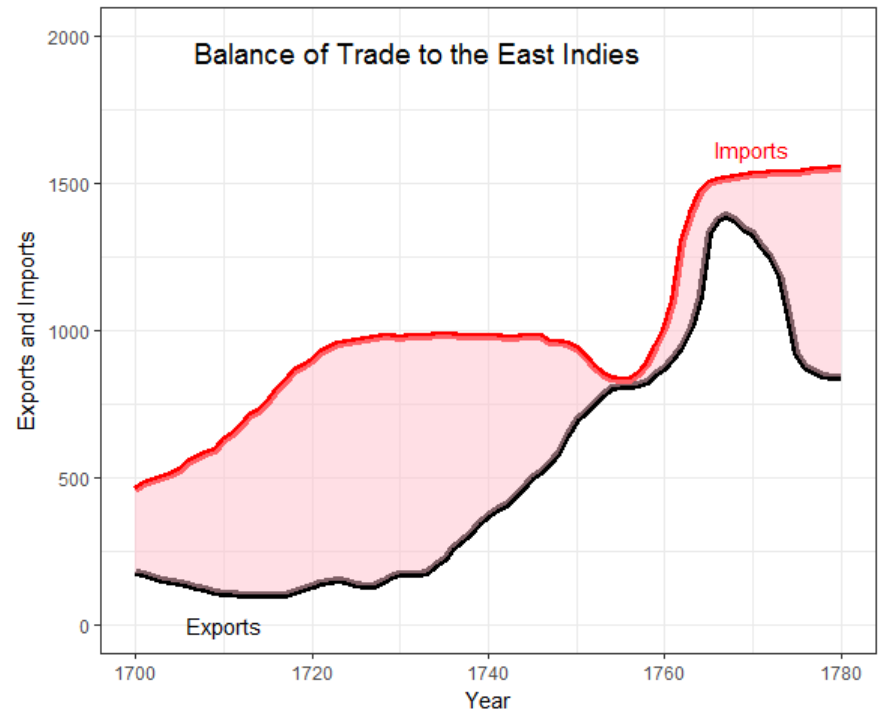
In the *Commercial and Political Atlas*, William Playfair used charts of imports and exports from England to its trading partners to ask “How are we doing”?

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
...	...	...	...

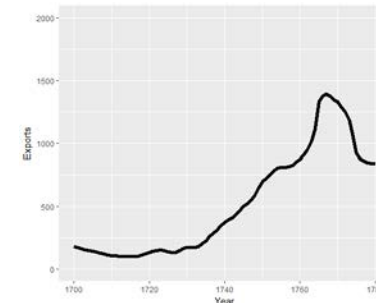


# ggplot thinking

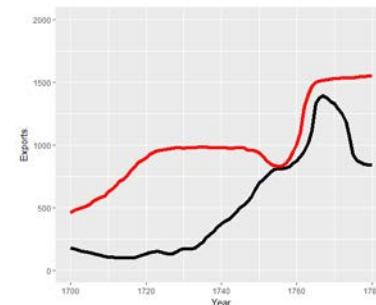
I want to plot two time series, & fill the area between them

Start with a line plot of Exports vs. Year: `geom_line()`

Add a layer for the line plot of Imports vs. Year

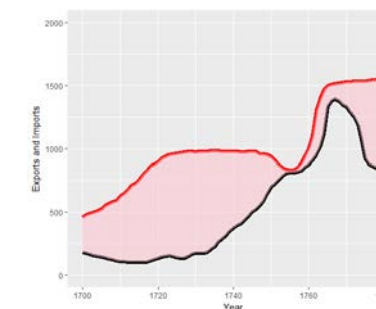


```
c1 <-  
ggplot(EastIndiesTrade, aes(x=Year, y=Exports)) +  
  ylim(0,2000) +  
  geom_line(colour="black", size=2) +  
  geom_line(aes(x=Year, y=Imports), colour="red", size=2)
```



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")
```

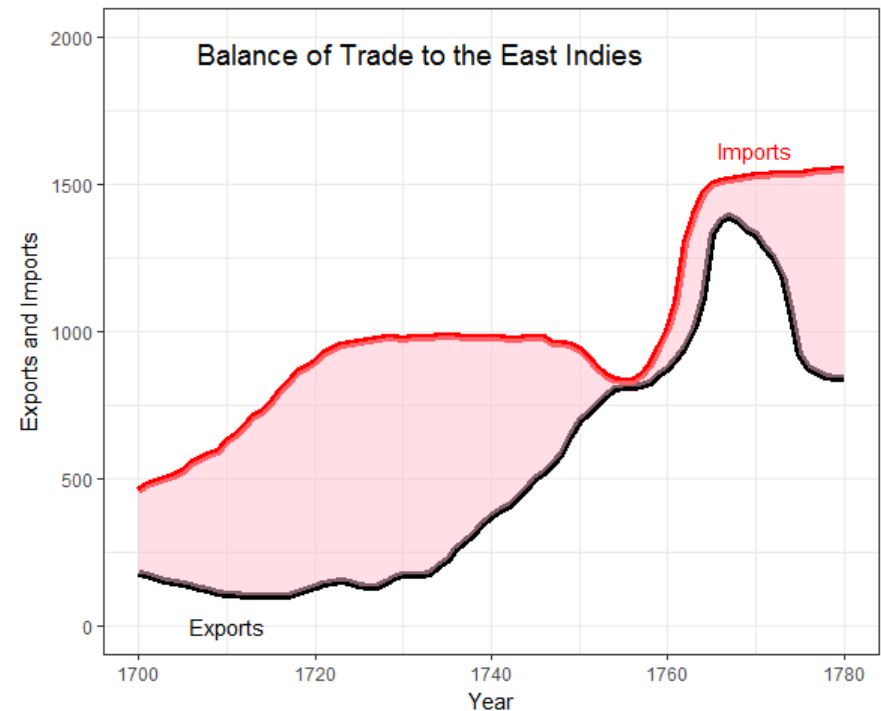


That looks pretty good. Add some text labels using `annotate()`

```
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

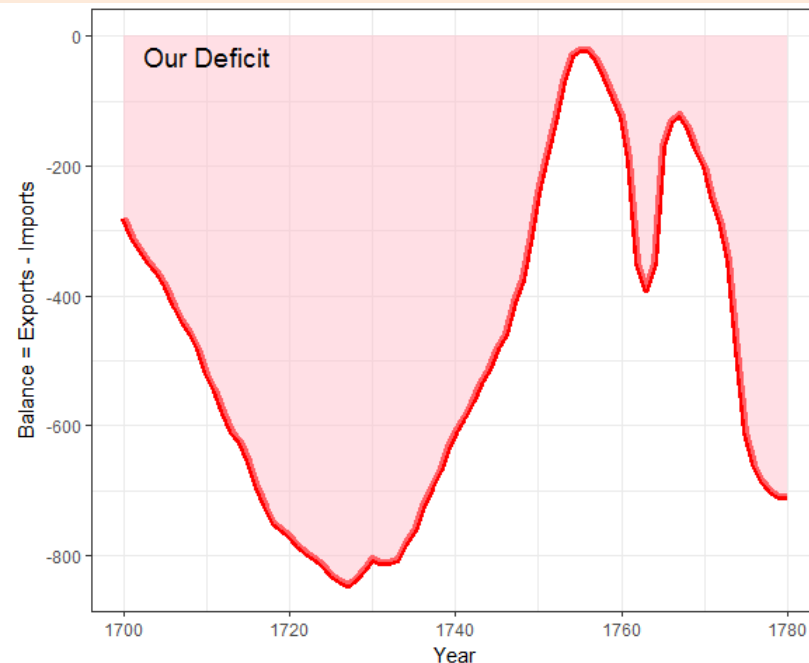
```
c1 <- c1 + theme_bw()
```



# 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()
```

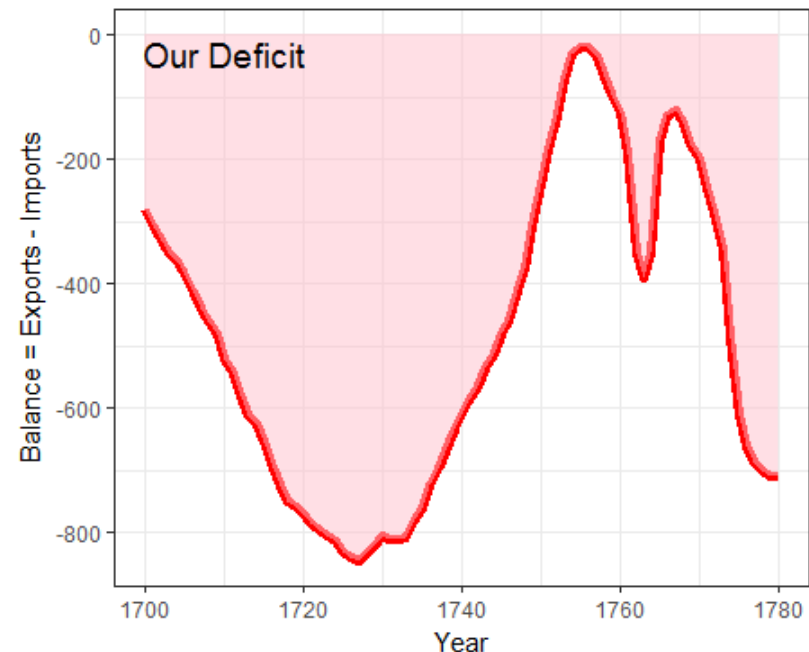
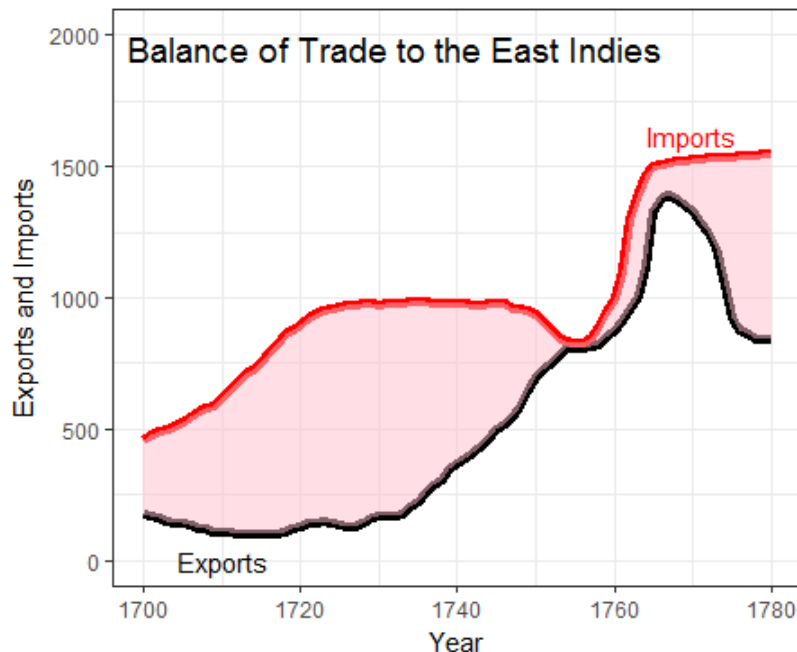


# Composing several plots

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")
```

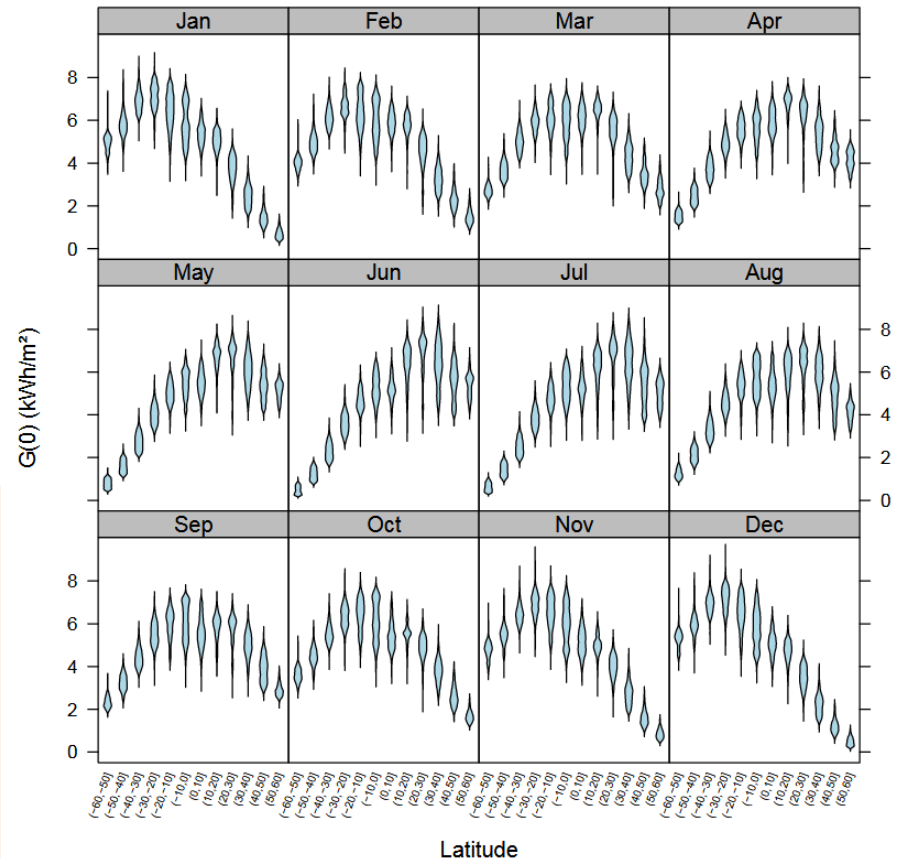


# Faceting & tidy data

Recall the lattice example plotting solar radiation vs. latitude over months of the year.

This was complicated, because the data structure was **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 ...
 $ 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 ...
 $ Feb: num 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 ...
 $ Apr: num 0 0 0 0 0 0 0 0 ...
 $ May: num 0 0 0 0 0 0 0 0 ...
 $ Jun: num 0 0 0 0 0 0 0 0 ...
 $ Jul: num 0 0 0 0 0 0 0 0 ...
 $ Aug: num 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 ...
 $ Oct: num 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 ...
 $ 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 ...
```



# tidying the data

In wide format, 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))
```

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]
```

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

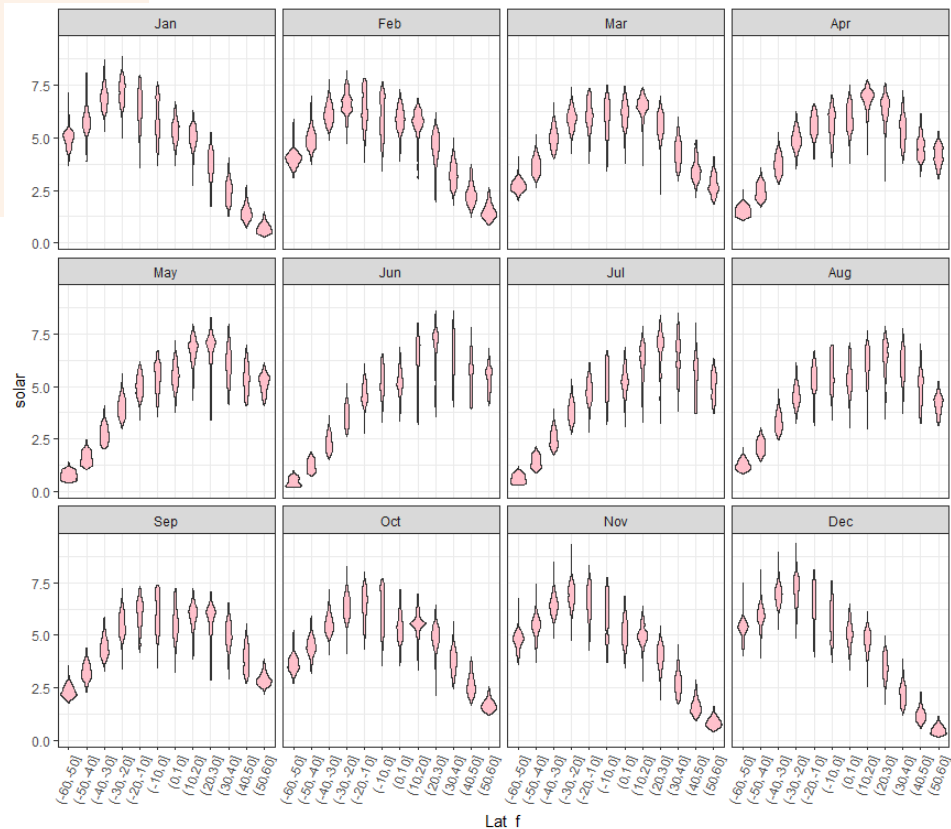
# plotting the tidy data

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

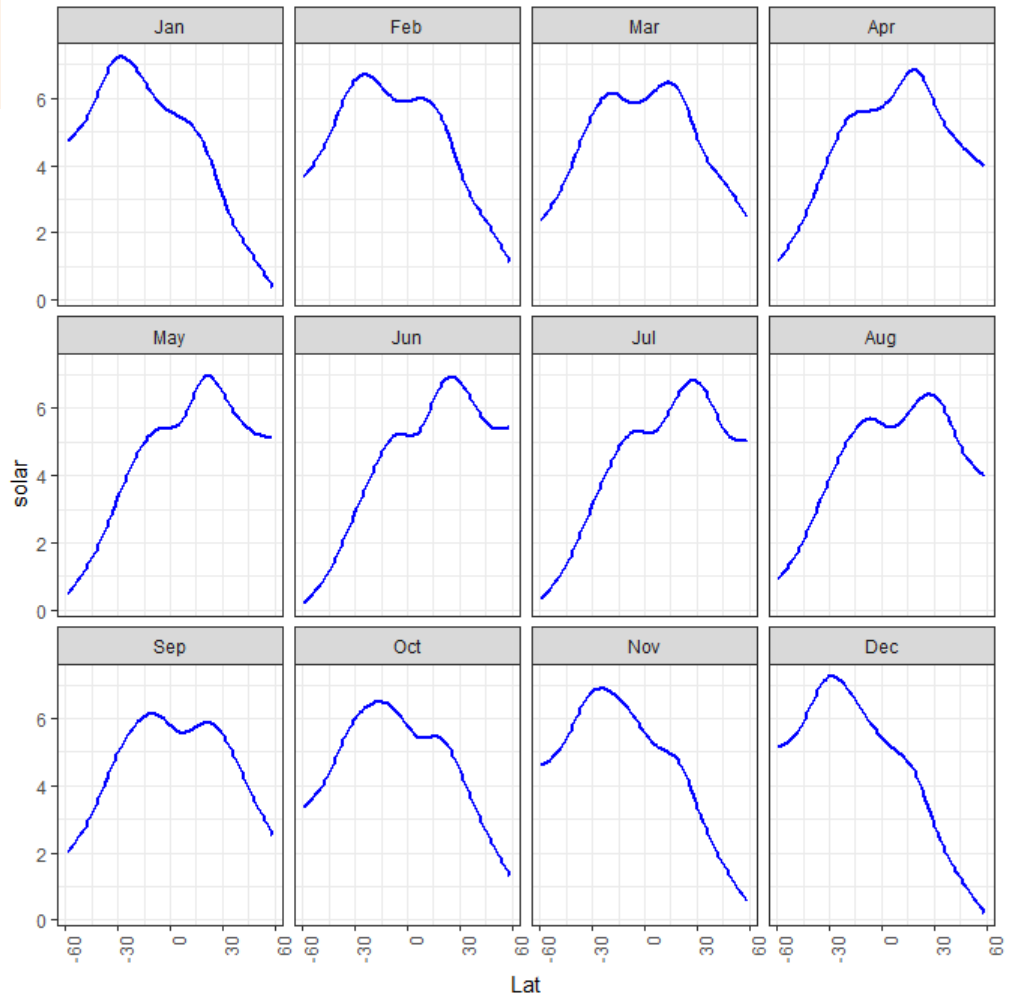


# plotting the tidy data: smoothing

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

Here we 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,  $s(\text{Lat})$

```
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.

Model terms:

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

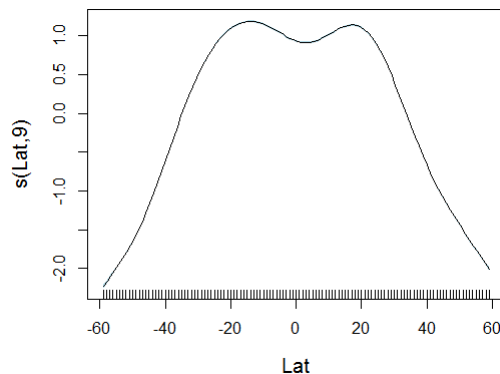
There are other model choices, but it is useful to visualize what we have done so far

# visualize the model

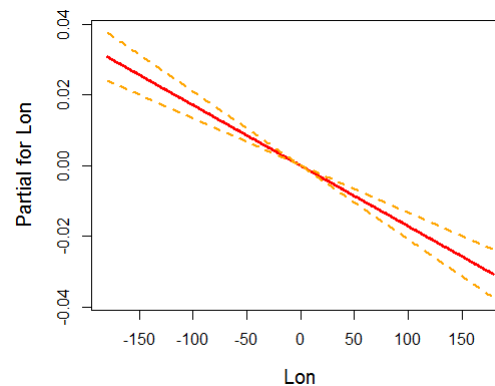
Effect plots show the fitted relationship between the response and model terms, averaged over other predictors.

The mgcv package has its own versions of these.

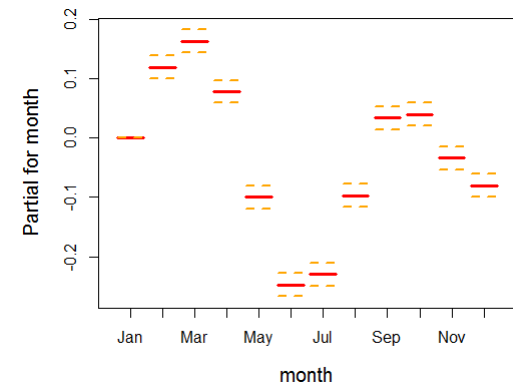
```
plot(nasa.gam, cex.lab=1.25)
termplot(nasa.gam, terms="month", se=TRUE, lwd.term=3, lwd.se=2, cex.lab=1.25)
termplot(nasa.gam, terms="Lon", se=TRUE, lwd.term=3, lwd.se=2, cex.lab=1.25)
```



why the dip at the equator?



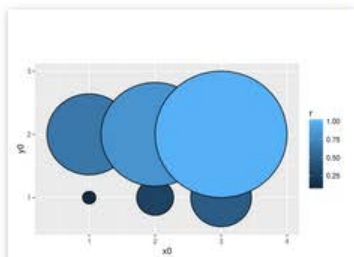
effect of longitude is very small, but maybe interpretable



month should be modeled as a time variable

# ggplot extensions

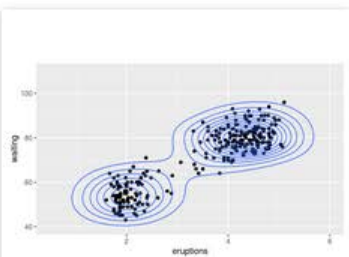
There are a large number of ggplot extensions. See: <http://www.ggplot2-exts.org/>



## ggforce

ggforce is aimed at providing missing functionality to ggplot2 through the extension system introduced with ggplot2 v2.0.0.

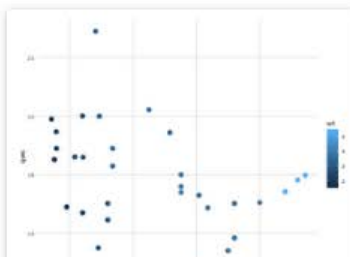
- author: thomasp85
- tags: visualization, general
- js libraries:



## ggalt

A compendium of 'geoms', 'coords' and 'stats' for 'ggplot2'.

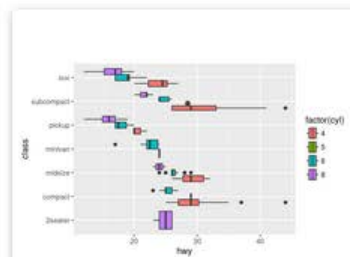
- author: hrbrmstr
- tags: visualization, general
- js libraries:



## ggiraph 67

htmlwidget to make 'ggplot' graphics interactive.

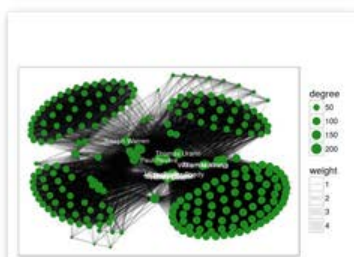
- author: davidgohel
- tags: visualization, general
- js libraries:



## ggstance

ggstance implements horizontal versions of common ggplot2 geoms.

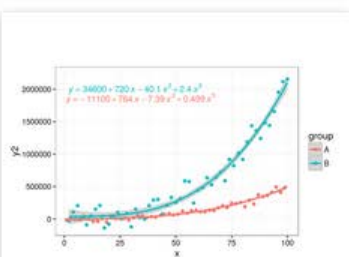
- author: lionel-
- tags: visualization, general
- js libraries:



## ggraph

ggraph is tailored at plotting graph-like data structures (graphs, networks, trees, hierarchies...).

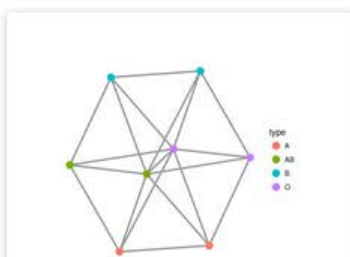
- author: thomasp85
- tags: visualization, general
- js libraries:



## ggpmisc

Miscellaneous Extensions to 'ggplot2'.

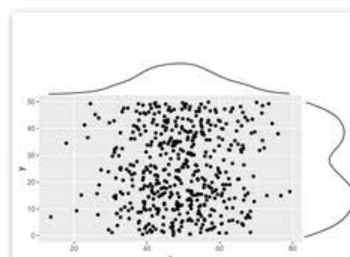
- author:
- tags: visualization, general
- js libraries:



## geomnet

geomnet implements network visualizations in ggplot2 via geom\_net.

- author: scytner
- tags: visualization, general
- js libraries:



## ggExtra

ggExtra lets you add marginal density plots or histograms to ggplot2 scatterplots.

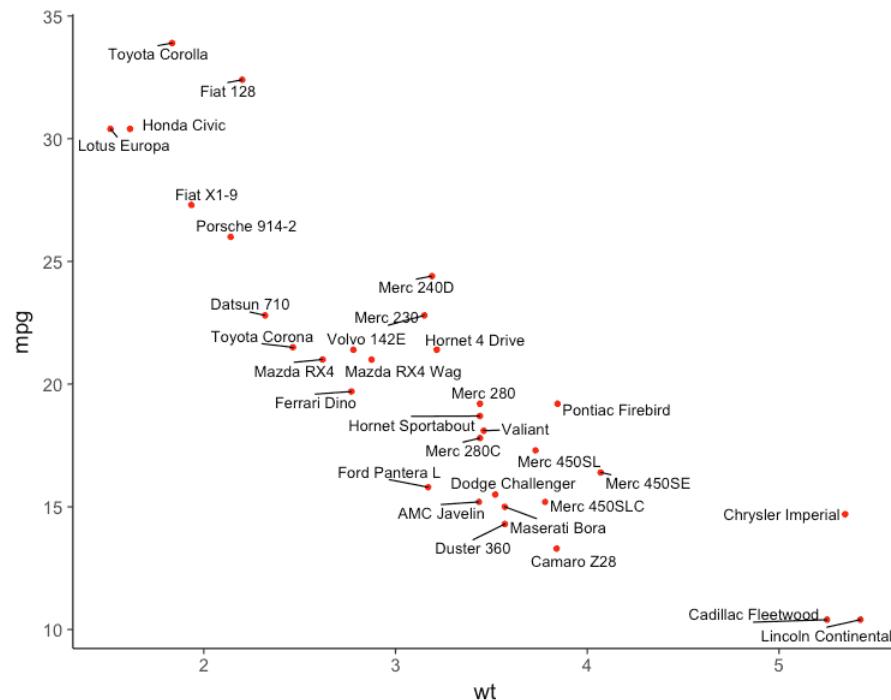
- author: daattali
- tags: histogram, marginal, density
- js libraries:



# ggplot extensions: ggrepel

```
devtools::install_github("slowkow/ggrepel")  
library(ggplot2)  
library(ggrepel)  
ggplot(mtcars, aes(wt, mpg)) +  
  geom_point(color = 'red') +  
  geom_text_repel(aes(label = rownames(mtcars))) +  
  theme_classic(base_size = 16)
```

Plotting text labels is often difficult  
ggrepel provides geoms for ggplot2 to repel overlapping text labels.

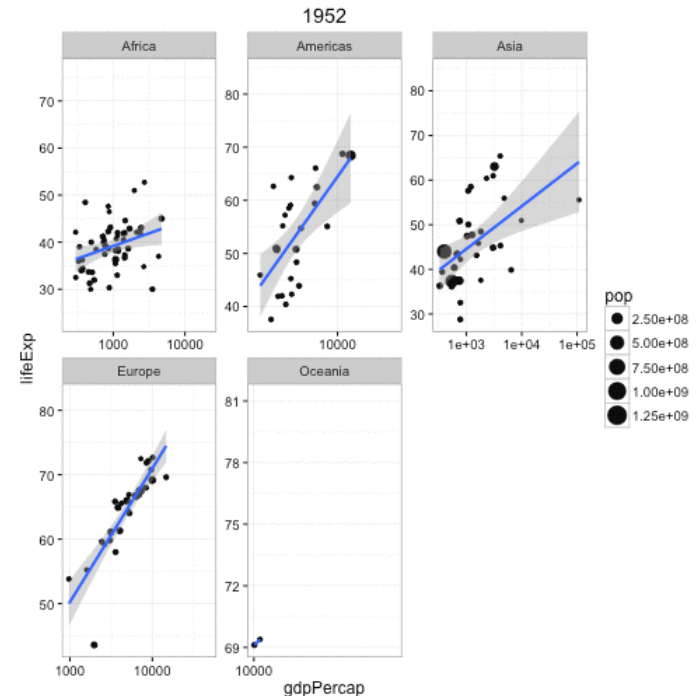


# ggplot extensions: gganimate

gganimate is a wrapper for the animation package with ggplot2.

It adds a **frame=** aesthetic, and animates the image as the frame variable changes

Install from github:  
`devtools::install_github("dgrtwo/gganimate")`



```
p5 <- ggplot(gapminder, aes(gdpPercap, lifeExp, size = pop, frame = year)) +  
  geom_point() +  
  geom_smooth(aes(group = year), method = "lm", show.legend = FALSE) +  
  facet_wrap(~continent, scales = "free") +  
  scale_x_log10()
```

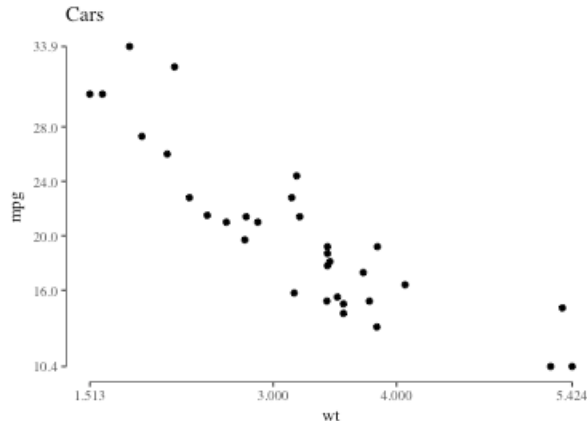
```
gganimate(p5)
```

# ggthemes

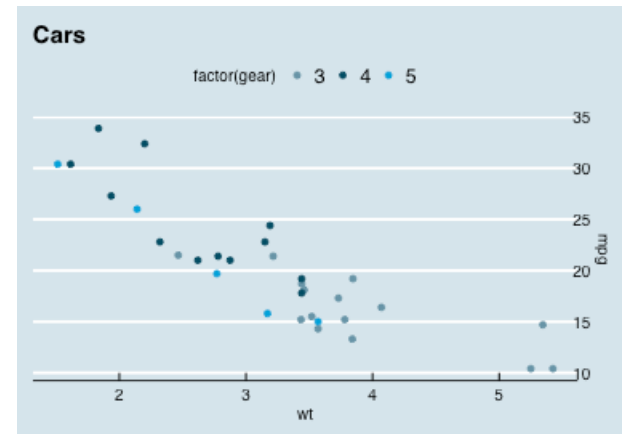
ggthemes provides a large number of extra  
geoms, scales, and themes for ggplot

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

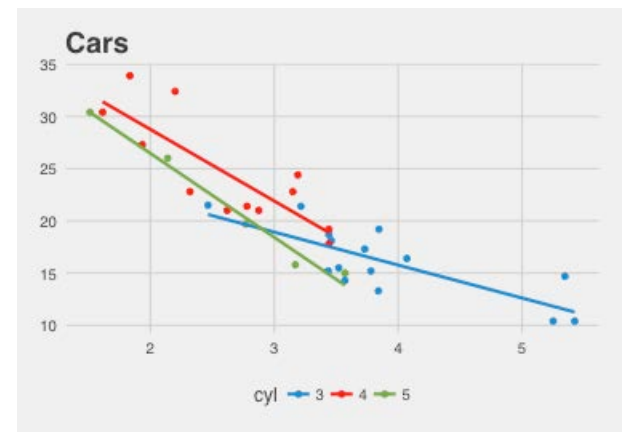
+ theme\_tufte()



+ theme\_economist()



+ 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

# Tables in R: xtable

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

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

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

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

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

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