SI 618
Exploratory Data Analysis

Factors, ggplot2, and smoothing
(+ SQL database access in R)

Instructor: Dr. Chris Teplovs (cteplovs@umich.edu)
Lead Developer, Digital Innovation Greenhouse, Office of Academic Innovation
Adjunct Lecturer of Information, School of Information
GSI: SungJin Nam (sjnam@umich.edu)

Course projects for 618

- There will be an individual course project
  - Worth 25% of course grade
- Focused on doing a more in-depth exploratory data analysis of a single or combined dataset
  - Propose multiple questions to explore in a dataset
  - OK if questions change a bit by the final report
- If you did a 601 project:
  - Should use a different or enhanced data source and methods
  - See me if you want to discuss alternatives
- Proposal, final report and presentation
- Presentations during last lecture (Friday, Dec. 16)

Coming up

- 1-page project proposal due
  Friday, November 18 1:00pm

SI 618 Data Exploration: Class Schedule

(Some curriculum details subject to change)

Date	Topic	Assignments Due
Week 1	Course introduction Basics of Programming with R	
Week 2	Basic analysis and visualization using ggplot2: qplot() Manipulating data frames using plyr	Homework 1
Week 3	Smoothing and Trend-finding. Building ggplot Layer by Layer, SQL	Homework 2
Week 4	Finding relationships between variables Time series and autocorrelation	Homework 3
Week 5	Clustering and Finding Outliers	Homework 4
Week 6	Factor Analysis Methods (PCA, EFA)	Homework 5
Week 7	Advanced topics **Project Presentations**	Project Due

Review from last week

- Basics of data frame manipulation
  - Filtering, transform, summarize (plyr)
- Basic visualization using qplot()
  - Summary statistics (histogram, boxplot)
  - Simple relationships between 2 variables (scatterplot)
  - Facets: conditioning on a third variable
  - Mapping variables to aesthetics
  - Geometric types: geom = "point", "line", …

Class Schedule for Today

- Factors in R
  - What are factors?
  - When and how to use factors?
- From qplot() to introducing ggplot()
- Smoothing
- Building plots layer by layer
- How to retrieve data from databases in R

Factors are categorical variables

- Factors take a limited number of potential values
- Factors have a corresponding set of strings called levels.
- Levels are used to display the factor's values

```
        > str(diamonds)
        'data.frame':      53940 obs. of  10 variables:
         $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
         $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
         $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5
...
```

See also:
http://www.statmethods.net/input/datatypes.html
http://statistics.ats.ucla.edu/stat/r/modules/factor_variables.htm

Examples of Factors

- *Nominal*: two or more categories, no intrinsic order
  - Male, Female
  - Restaurant cuisines: Chinese, Italian, Japanese …
- *Ordinal*: Ordered categories (ordered factor)
  - Rating scale of 1 to 5 (stars)
    - Low < Medium < High
    - Tall < Average < Short
    - Months

See also:
http://www.statmethods.net/input/datatypes.html
http://statistics.ats.ucla.edu/stat/r/modules/factor_variables.htm

Factor example: mtcars dataset

```
> str(mtcars)
'data.frame':  32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Factor example

```
> summary(newmt)
      mpg             cyl             disp             hp       
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
 Median :19.20   Median :6.000   Median :196.3   Median :123.0  
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
      drat             wt             qsec      
 Min.   :2.760   Min.   :1.513   Min.   :14.50  
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89  
 Median :3.695   Median :3.325   Median :17.71  
 Mean   :3.597   Mean   :3.217   Mean   :17.85  
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90  
 Max.   :4.930   Max.   :5.424   Max.   :22.90  
       vs               am              gear      
 Min.   :0.0000   Min.   :0.0000   Min.   :3.000  
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:3.000  
 Median :0.0000   Median :0.0000   Median :4.000  
 Mean   :0.4375   Mean   :0.4062   Mean   :3.688  
 3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:4.000  
 Max.   :1.0000   Max.   :1.0000   Max.   :5.000  
      carb      
 Min.   :1.000  
 1st Qu.:2.000  
 Median :2.000  
 Mean   :2.812  
 3rd Qu.:4.000  
 Max.   :8.000  
```

Creating factors with `factor()` and manipulating levels with `levels()`

```
> fgear = factor(mtcars$gear)
> fgear
 [1] 4 4 4 3 3 3 3 4 4 4 4 3 3 3 3 3 3 4 4 4 3 3 3 3 3 4 5 5 5 5 5 4
Levels: 3 4 5
> fgear = factor(mtcars$gear, labels = c("three", "four", "five"))
> fgear
 [1] four  four  four  three three three three four  four  four  four  three three three three three three four  four  four  three
three three three three four  five  five  five
[30] five  five  four
Levels: three four five
> levels(fgear) = c('three','four','five')
> levels(fgear)
[1] "three" "four"  "five"
> nlevels(fgear)
[1] 3
> newmt = data.frame(mtcars, fgear)
> str(newmt)
'data.frame':  32 obs. of  12 variables:
 $ mpg  : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl  : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp : num  160 160 108 258 360 ...
 $ hp   : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat : num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt   : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec : num  16.5 17 18.6 19.4 17 ...
 $ vs   : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am   : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear : num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb : num  4 4 1 1 2 1 4 2 2 4 ...
 $ fgear: Factor w/ 3 levels "three","four",..: 2 2 2 1 1 1 1 2 2 2 ...
```

Factor example

```
> summary(newmt)
      mpg             cyl             disp             hp
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
 Median :19.20   Median :6.000   Median :196.3   Median :123.0
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
      drat             wt             qsec
 Min.   :2.760   Min.   :1.513   Min.   :14.50
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89
 Median :3.695   Median :3.325   Median :17.71
 Mean   :3.597   Mean   :3.217   Mean   :17.85
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90
 Max.   :4.930   Max.   :5.424   Max.   :22.90
       vs               am              gear
 Min.   :0.0000   Min.   :0.0000   Min.   :3.000
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:3.000
 Median :0.0000   Median :0.0000   Median :4.000
 Mean   :0.4375   Mean   :0.4062   Mean   :3.688
 3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:4.000
 Max.   :1.0000   Max.   :1.0000   Max.   :5.000
      carb        fgear
 Min.   :1.000   three:15
 1st Qu.:2.000   four :12
 Median :2.000   five : 5
 Mean   :2.812
 3rd Qu.:4.000
 Max.   :8.000
```

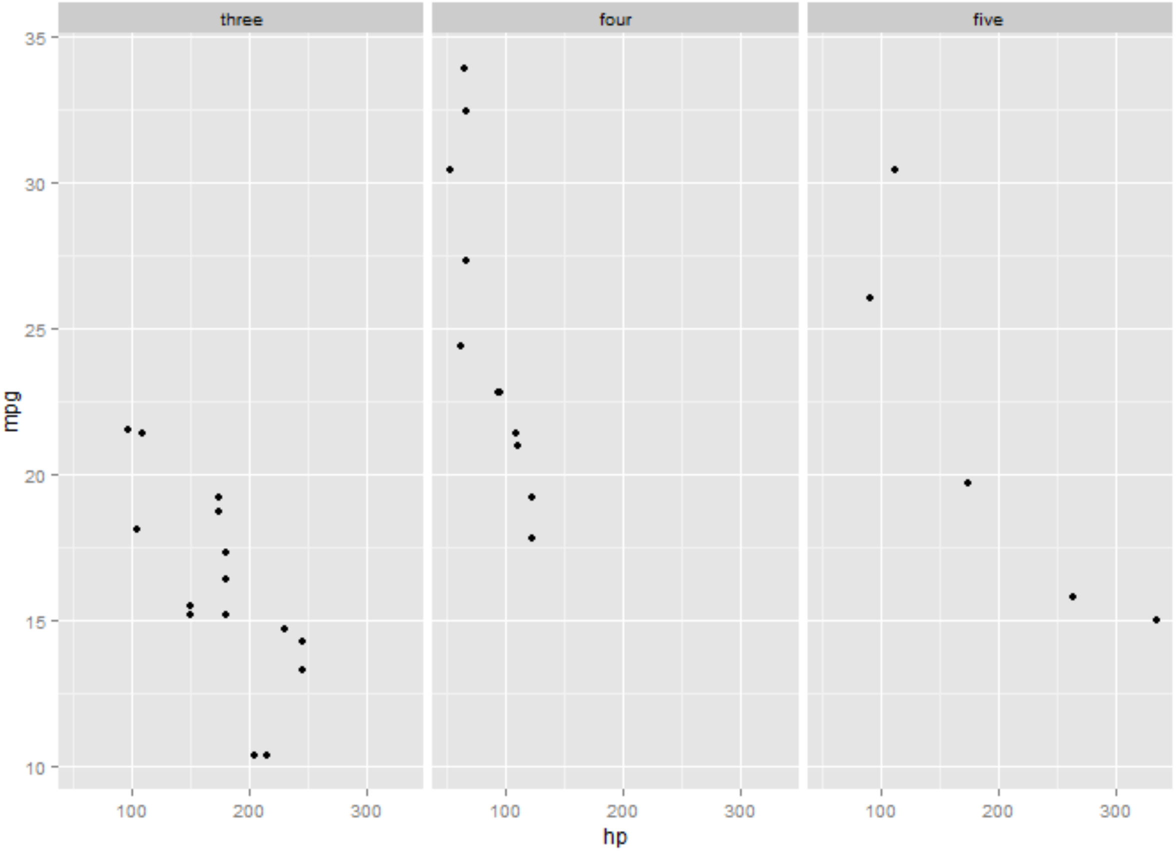
What's the relationship between a car's engine horsepower and its fuel economy (miles-per-gallon)? Per gear level?

```
qplot(hp, mpg, facets=~gear, data=mtcars)
```

```
Error in layout_base(data, vars, drop = drop) :
  At least one layer must contain all variables used for facetting
```

What's the relationship between a car's engine horsepower and its fuel economy (miles-per-gallon)? Per gear level?

```
qplot(hp, mpg, facets=~fgear, data=newmt)
```



© 2016 Chris Teplovs

Ordered factors

- Ordered factors allow comparisons between values
- Pass `ordered = TRUE` to create an ordered factor

```
> fgear[1] > fgear[4]
[1] NA
Warning message:
In Ops.factor(fgear[1], fgear[4]) : > not meaningful for factors

> fgear = factor(mtcars$gear, labels = c("three", "four", "five"),
ordered=TRUE)
> fgear[1]
[1] four
Levels: three < four < five
> fgear[4]
[1] three
Levels: three < four < five
> fgear[1] > fgear[4]
[1] TRUE
```

But how would you create factors from floating-point variables? Answer: cut() function to the rescue!

Suppose we want to categorize cars into 3 speeds using the qsec time.

```
> factor.qsec <- cut(mtcars$qsec, 3)
> factor.qsec
 [1] (14.5,17.3] (14.5,17.3] (17.3,20.1]…
Levels: (14.5,17.3] (17.3,20.1] (20.1,22.9]
> factor.qsec <- cut(mtcars$qsec, 3, labels=c("fast", 
"medium", "slow"))
> factor.qsec
 [1] fast   fast   medium medium fast   slow   fast   
medium slow   medium medium medium medium medium medium
medium medium medium medium medium medium fast   medium
fast   fast  
[26] medium fast   fast   fast   fast   fast   medium
Levels: fast medium slow
```

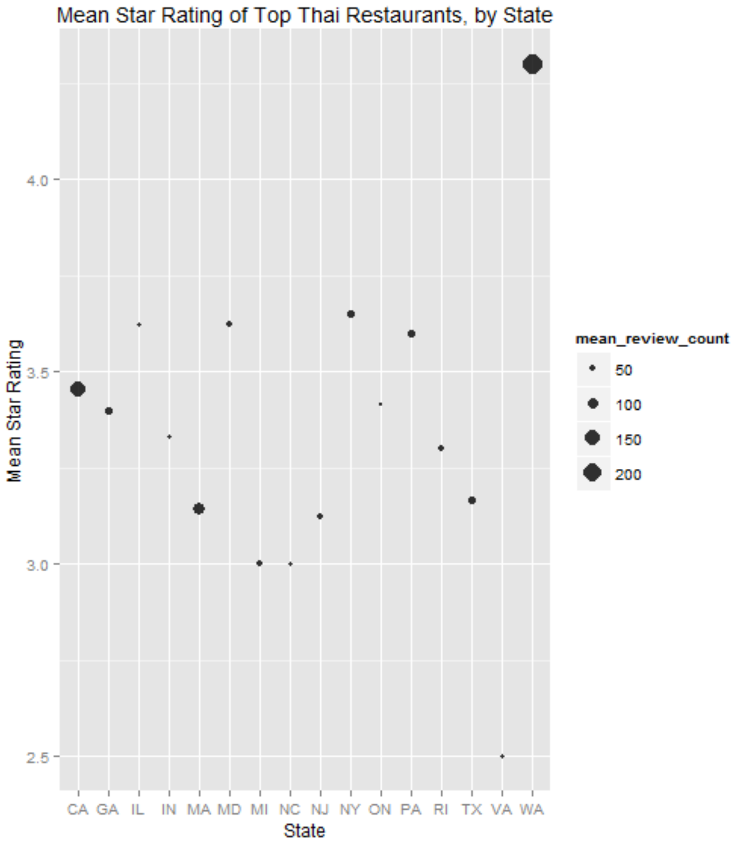
What if we want to create new factors out of combinations of existing ones? Answer: the interaction() function at your service!

- Can be useful to create new factors for all combinations of two existing factors
  - e.g. categorize cars by (gear, cylinder) combination
- Default: include all possible combinations
- Retain interactions only for combinations actually seen in the data: drop = TRUE

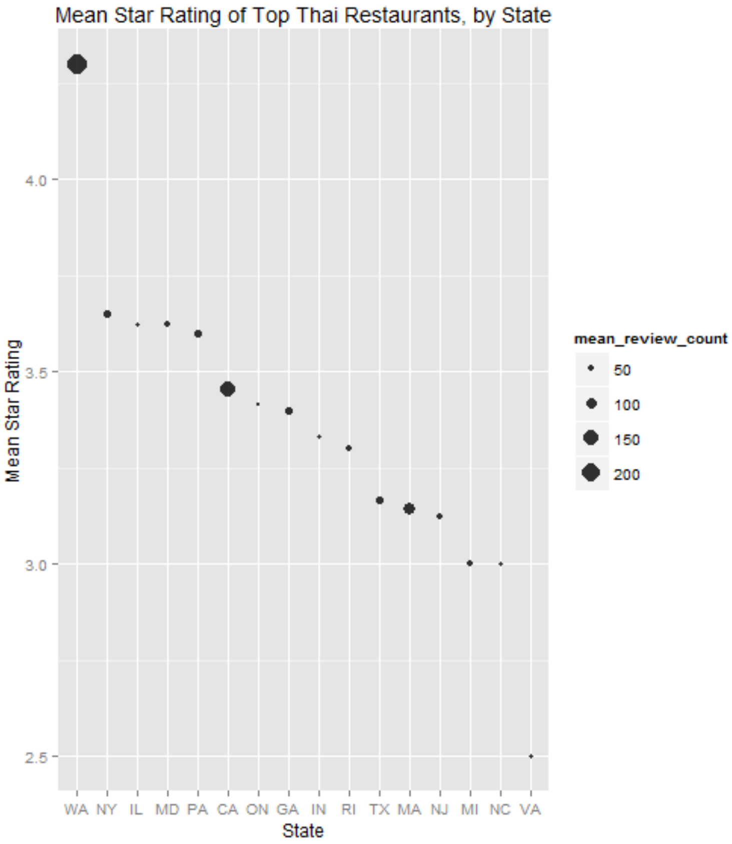
```
> interaction(mtnew$gear, mtnew$cyl)
[1] 4.6 4.6 4.4 3.6 3.8 3.6 3.8 4.4 4.4 4.6 4.6 3.8 3.8 3.8 3.8 3.8 3.8
4.4 4.4 4.4 3.4 3.8 3.8 3.8 3.8 4.4 5.4 5.4 5.8 5.6 5.8 4.4
Levels: 3.4 4.4 5.4 3.6 4.6 5.6 3.8 4.8 5.8
> interaction(mtnew$gear, mtnew$cyl, drop = TRUE)
[1] 4.6 4.6 4.4 3.6 3.8 3.6 3.8 4.4 4.4 4.6 4.6 3.8 3.8 3.8 3.8 3.8 3.8
4.4 4.4 4.4 3.4 3.8 3.8 3.8 3.8 4.4 5.4 5.4 5.8 5.6 5.8 4.4
Levels: 3.4 4.4 5.4 3.6 4.6 5.6 3.8 5.8
```

Reordering factors for sorted plots

```
qplot(state, mean_stars, ...)
```

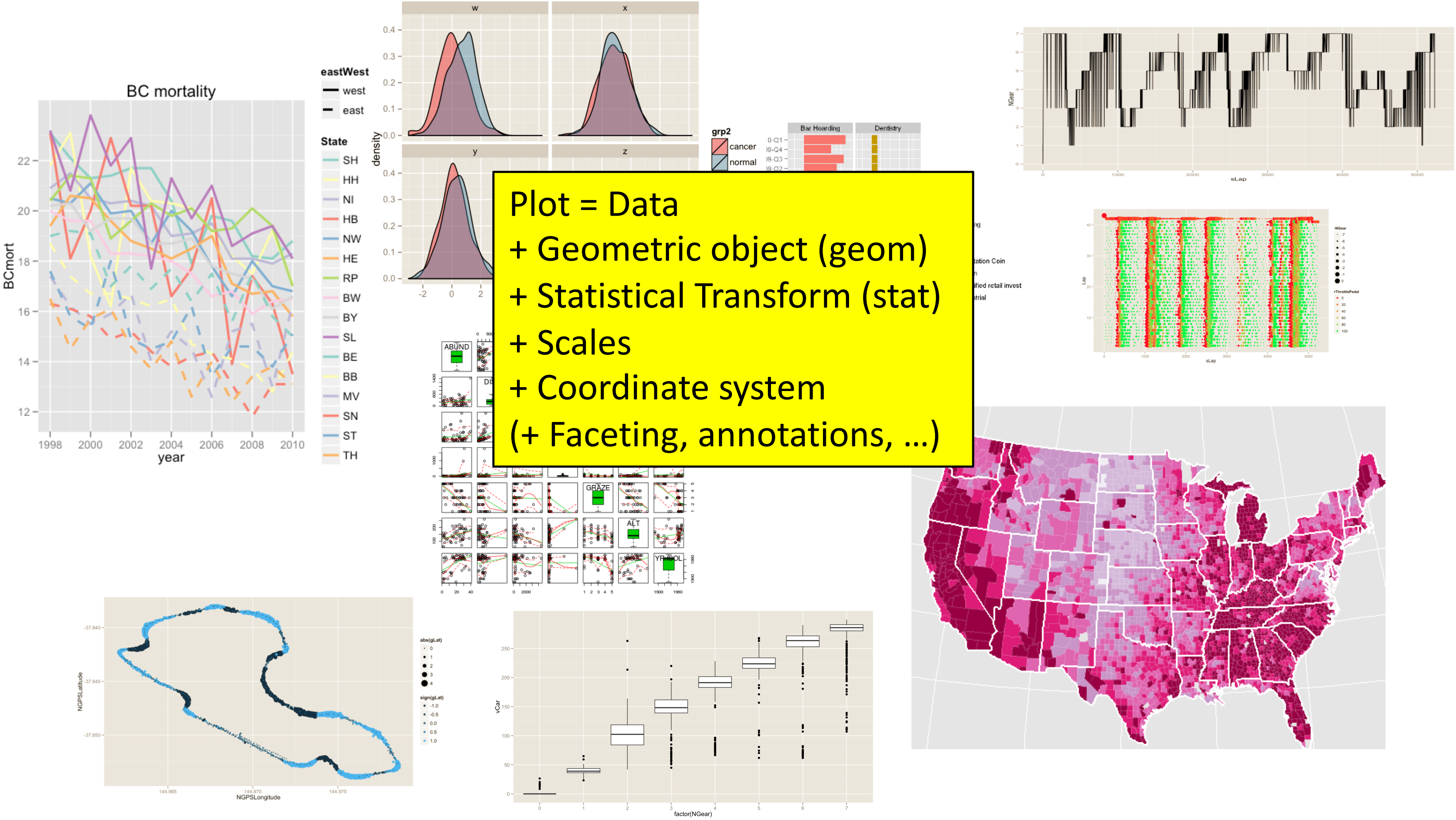


```
qplot(reorder(state, -mean_fields$mean_stars),
    mean_stars, ...)
```



Beyond qplot:
The power of ggplot()

These wildly different plots share a common language: 'The grammar of graphics'



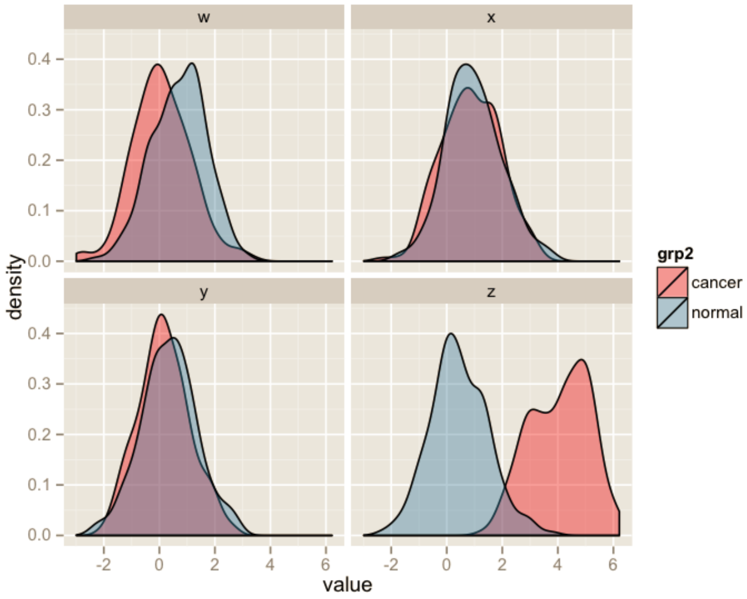
Plot = Data
+ Geometric object (geom)
+ Statistical Transform (stat)
+ Scales
+ Coordinate system
(+ Faceting, annotations, …)

ggplot2 design

- The ggplot2 package, created by Hadley Wickham, offers a powerful graphics language for creating elegant and complex plots.
- Provides a set of building blocks
- That can be put together creatively in many different ways
- To build rich visualizations w/ compact syntax

What is a graphic?

- Mapping from data to
  - *aesthetic* attributes (color, size, shape)
  - of *geometric* objects (points, lines, bars)
  - that may also *transform* the data
  - and drawn on a specific *coordinate* system
- Faceting:
  - Same plot, different subsets of the data

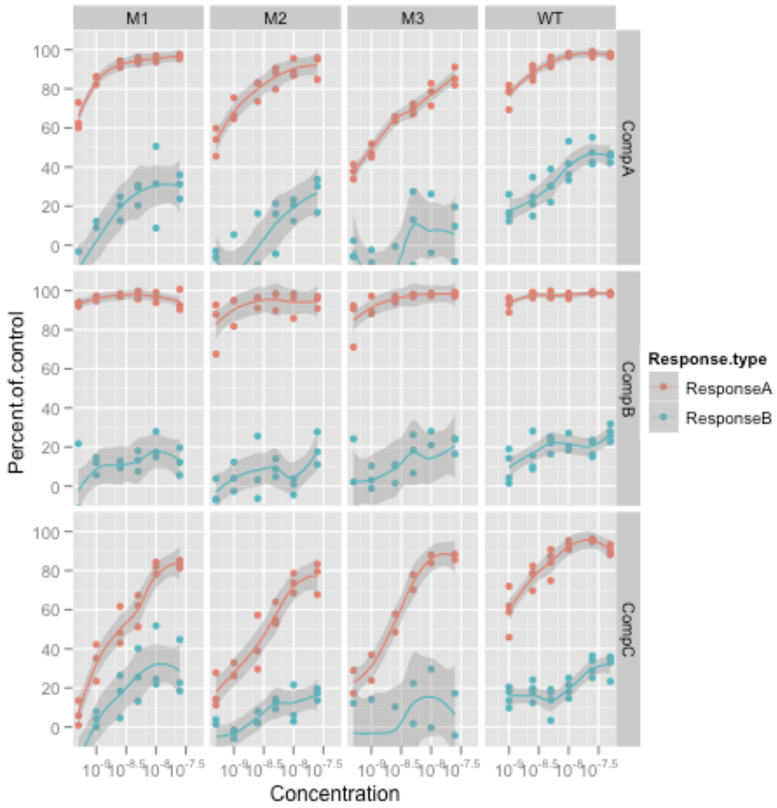


What ggplot2 does <u>not</u> do

- Tell you what graphic to create
- Specify exhaustive details of display
- Tell you how to make the graph visually pleasing
- Provide interactive exploration

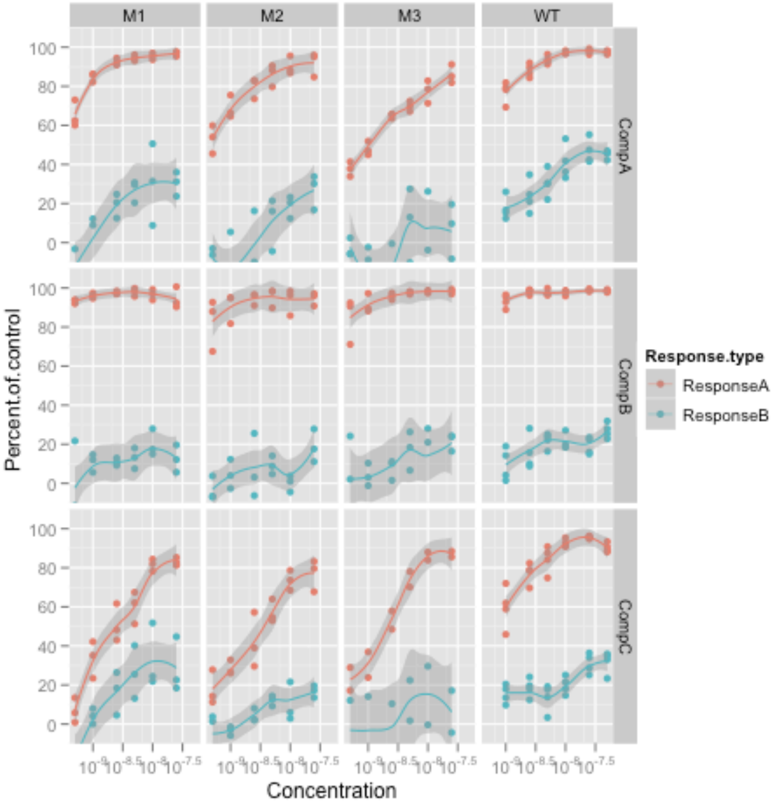
Most plots can be built from a common 'grammar' of graphical building blocks (ggplot basics)

- Data
  - Data frame: rows (records) and columns (variables)
- Aesthetics (aes)
  - Maps variables to graph attributes. A variable may control where points appear, the color or shape of a point, the height of a bar and so on.
- Geometric object (geom)
  - The geometric objects. Bars, points, lines, …

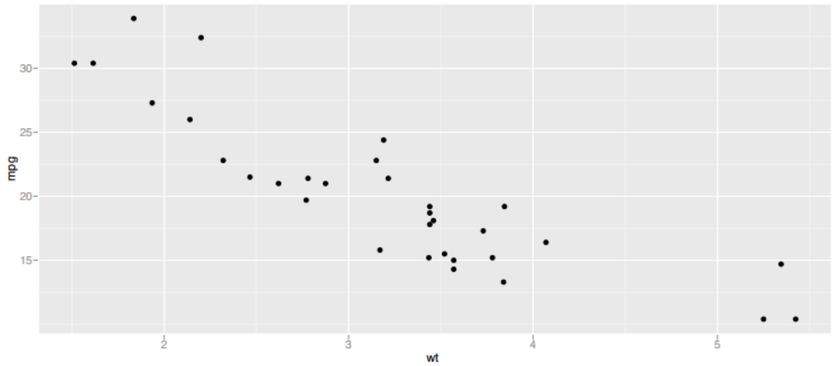


Most plots can be built from a common 'grammar' of graphical building blocks (ggplot basics)

- Statistical transform (stat)
  - Functions you apply to the data, like smoothing or linear regression you might need to draw a line or fitted curve.
- Scales
  - Legends that show things like circle = male, square = female.
- Coordinate system
  - How to map values to a 2-d display surface
- (+ position adjustment, faceting, guides, annotations)



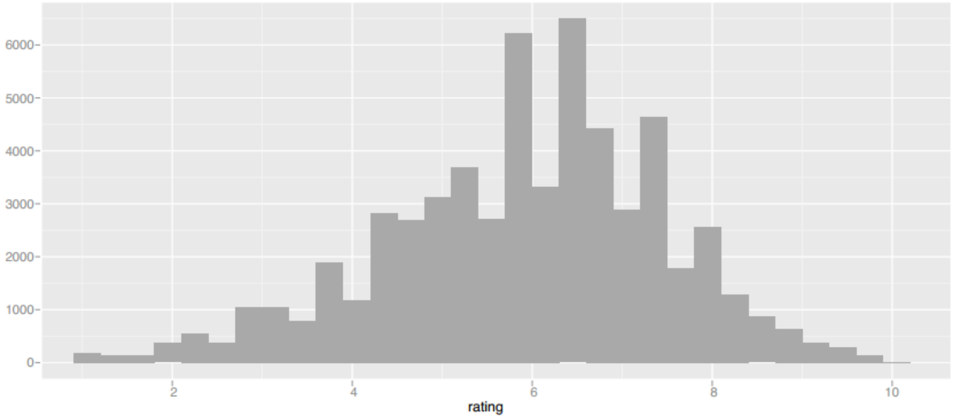
Scatterplot



- Geom: point
- Stat: identity
- Scale: linear
- Coordinate system: Cartesian

Source: http://ggplot2.org/resources/2007-vanderbilt.pdf

Histogram



- Geom: bar
- Stat: bin
- Scale: linear
- Coordinate system: Cartesian

Source: http://ggplot2.org/resources/2007-vanderbilt.pdf

Three ways to call ggplot()

1. All layers use the same data and the same set of aesthetics

   Can also be used to add a layer using data from another data frame.

   `ggplot(df, aes(x, y, ...))`

   - New layers inherit the aesthetics, mappings

2. Default data frame to use for the plot, but no aesthetics
   Useful when one data frame is used predominantly as layers are added, but the aesthetics may vary from one layer to another.
   `ggplot(df)`

3. Skeleton ggplot object which is fleshed out as layers are added.
   Useful when multiple data frames are used to produce different layers, as is often the case in complex graphics.
   `ggplot()`

Source: http://docs.ggplot2.org/0.9.3.1/ggplot.html

Calling ggplot

`ggplot``(`*`data, aesthetic mapping`*`)`

`(+ stat) (+ geom) (+ position)` …

Calling ggplot

```
ggplot(data, aesthetic mapping)
(+ stat) (+ geom) (+ position) …
```

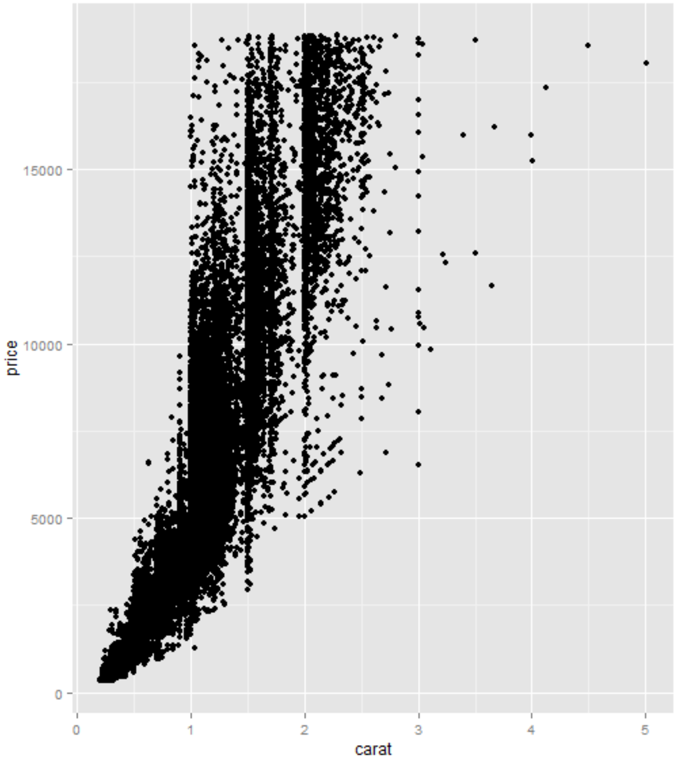
```
d <- ggplot(diamonds,
        aes(x=carat, y=price))
```

Error: No layers in plot

Calling ggplot

```
ggplot(data, aesthetic mapping)
(+ stat) (+ geom) (+ position) …
```

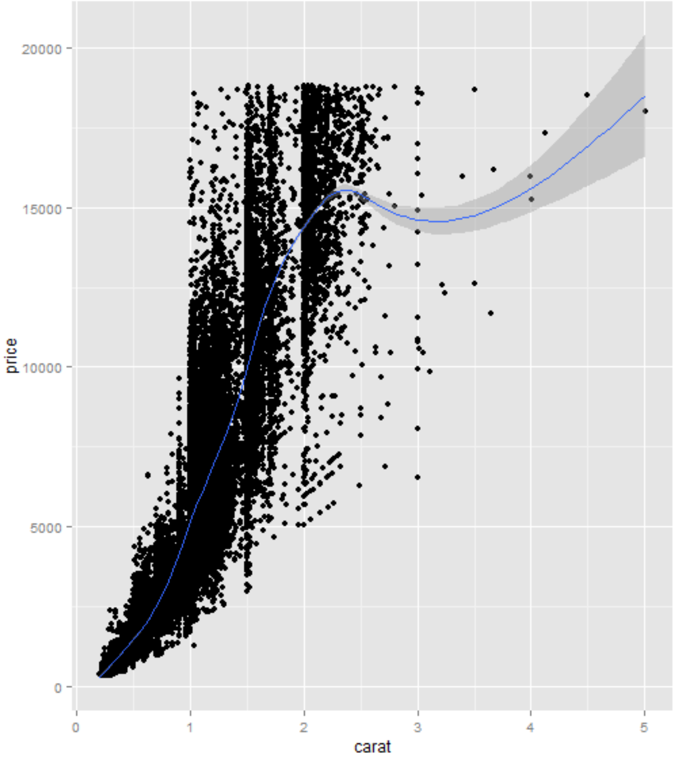
```
d <- ggplot(diamonds,
         aes(x=carat, y=price))
     + geom_point()
```



Calling ggplot

```
ggplot(data, aesthetic mapping)
(+ stat) (+ geom) (+ position) …
```

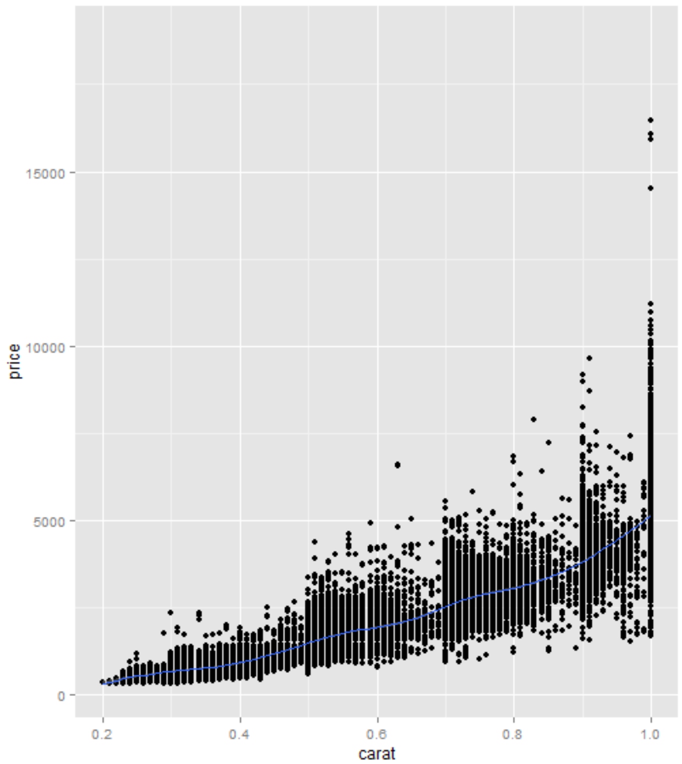
```
d <- ggplot(diamonds,
            aes(x=carat, y=price))
     + geom_point()
     + stat_smooth()
```



Calling ggplot

```
ggplot(data, aesthetic mapping)
(+ stat) (+ geom) (+ position) …
```

```
d <- ggplot(diamonds,
            aes(x=carat, y=price))
     + geom_point()
     + stat_smooth()
     + xlim(0.2, 1)
```



Aesthetic attributes: color, size, shape, alpha…

- Visual properties that affect how the data are displayed
- Each aesthetic can be mapped to a variable
  - Color and shape work well with categorical variables
  - Size works better for continuous variables
- Every aesthetic attribute has a *scale*
  - A scale maps data values to aesthetic values
  - ggplot will add a legend automatically if needed

Plots convey information via aspects of their aesthetics

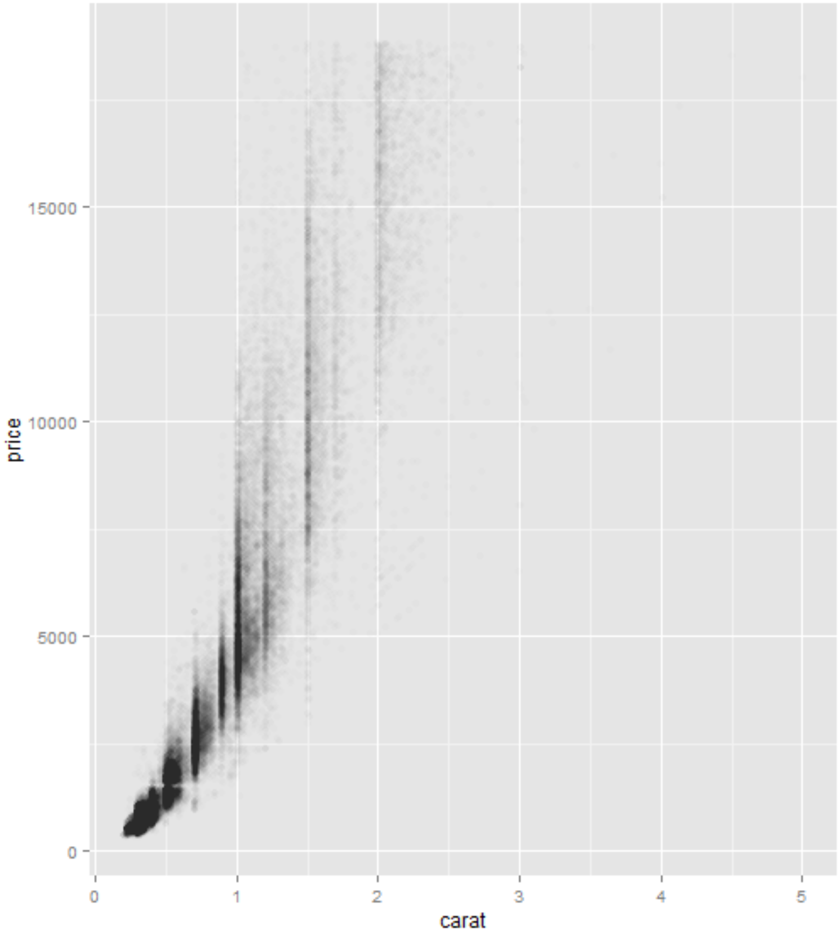
Some aesthetics that plots use are:

- x position
- y position
- size of elements
- shape of elements
- color of elements

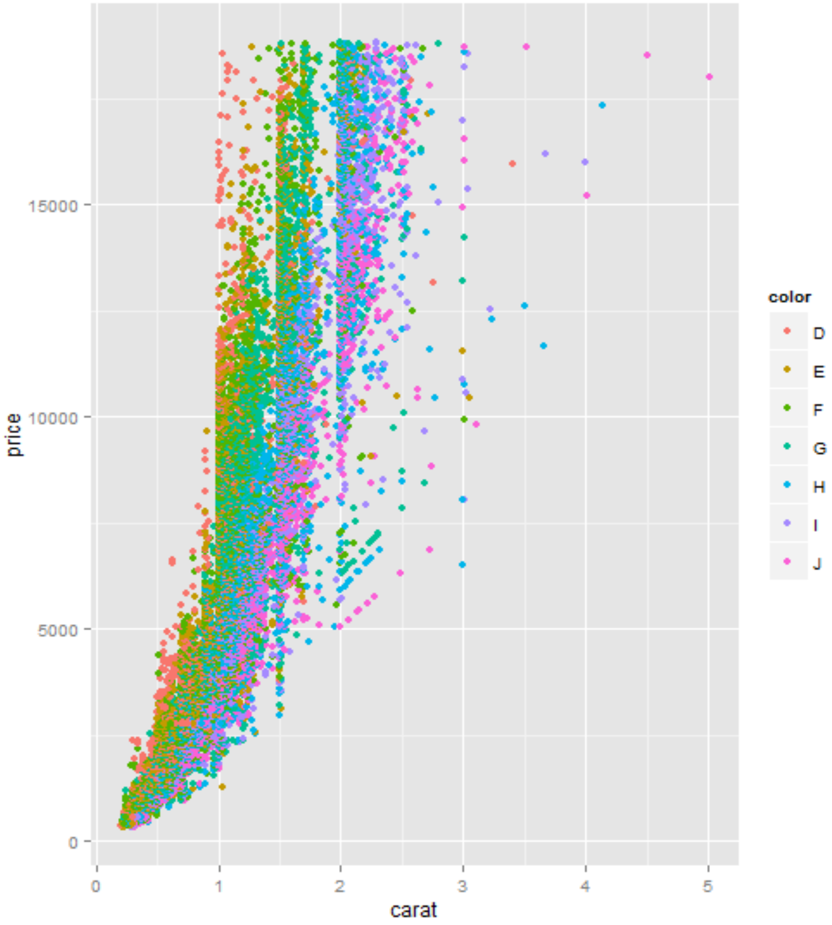
The elements in a plot
are **geometric** shapes, like

- points
- lines
- line segments
- bars
- text

```
ggplot(diamonds, aes(carat, price))
  + geom_point(alpha = I(1/100))
```

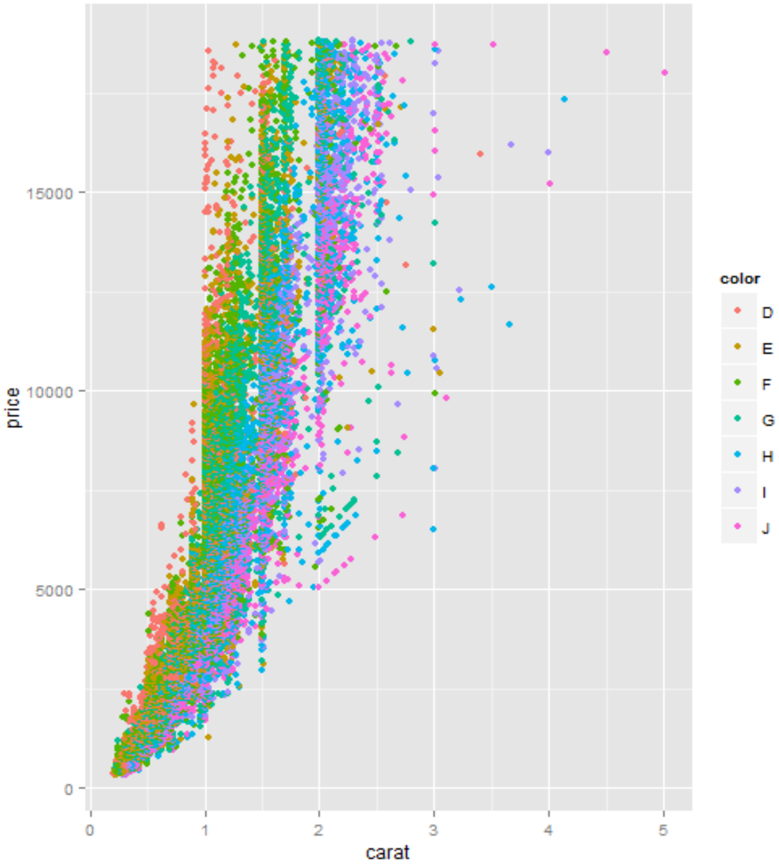


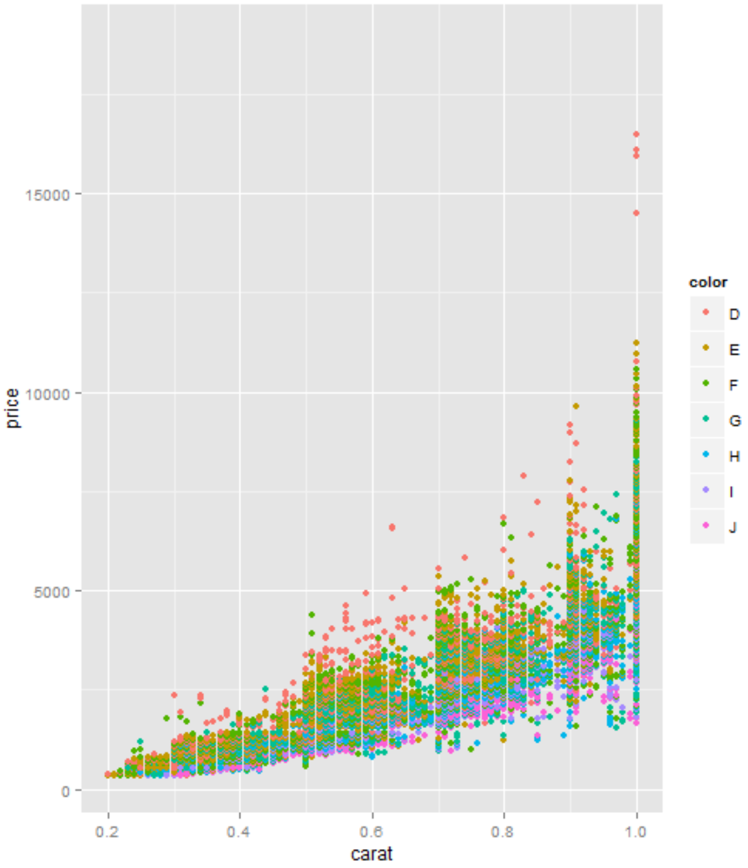
```
ggplot(diamonds, aes(carat, price, colour = color))
+ geom_point()
```



Use '+' to add new features to a plot and last_plot() to update previous result

```
last_plot() + xlim(0.2, 1)
```





© 2016 Chris Teplovs

Some geoms have their own aesthetics

- points
  - point shape
  - point size
- lines
  - line type
  - line weight
- bars
  - y minimum
  - y maximum
  - fill color
  - outline color
- text
  - label value

See Wickham Chapter 4, Table 4.2 for a complete geom list

Geometric objects in ggplot: `geom="…"`

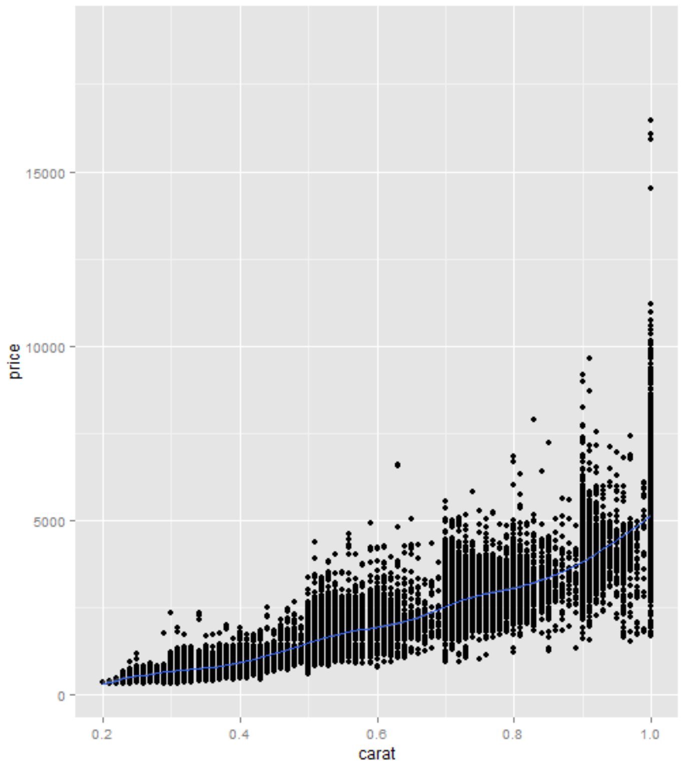
The "geom" type controls the plot you see. We can produce different plots by varying the "geom" geometry type. Three basic "geom":

- point
- jitter
- boxplot

Calling ggplot

```
ggplot(data, aesthetic mapping)
(+ stat) (+ geom) (+ position) …
```

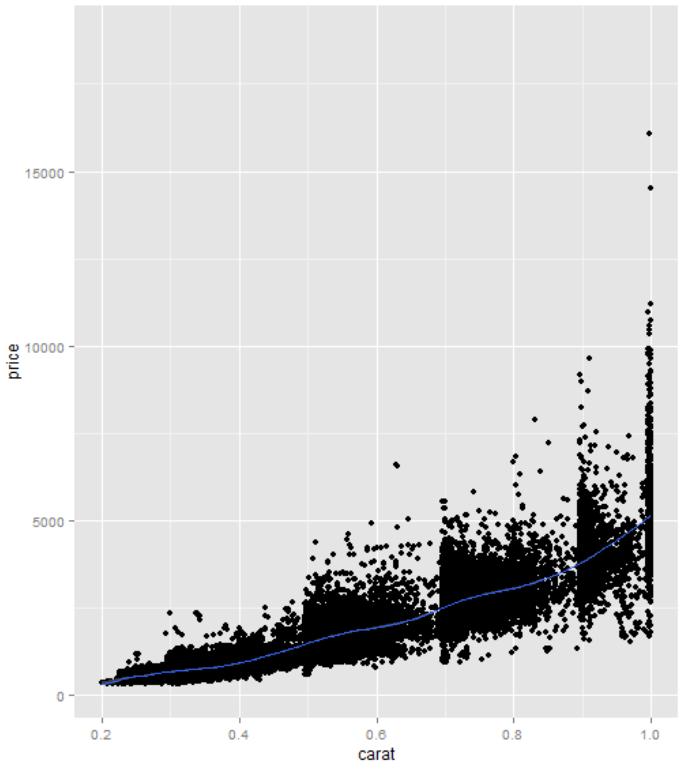
```
d <- ggplot(diamonds,
            aes(x=carat, y=price))
     + geom_point()
     + stat_smooth()
     + xlim(0.2, 1)
```



Calling ggplot

```
ggplot(data, aesthetic mapping)
(+ stat) (+ geom) (+ position) …
```

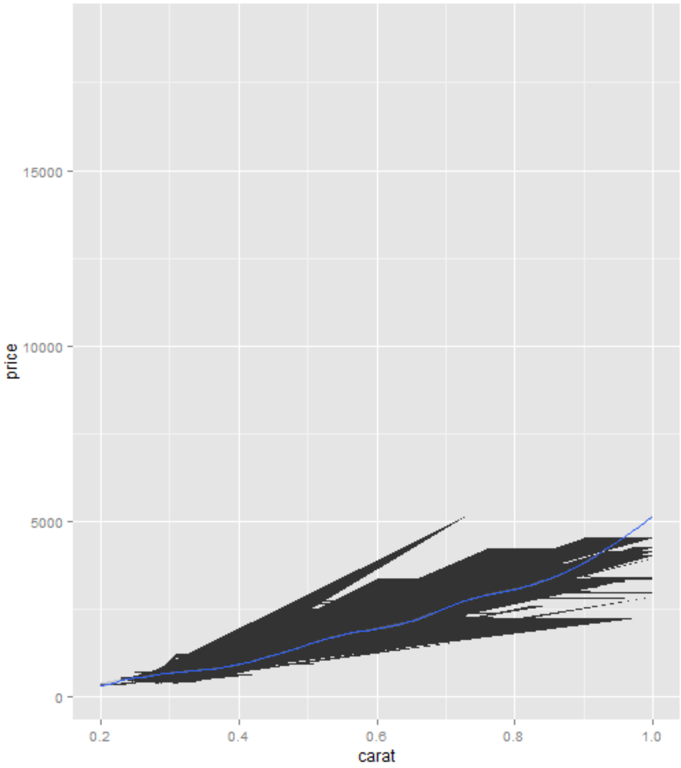
```
d <- ggplot(diamonds,
            aes(x=carat, y=price))
     + geom_jitter()
     + stat_smooth()
     + xlim(0.2, 1)
```



Calling ggplot

```
ggplot(data, aesthetic mapping)
(+ stat) (+ geom) (+ position) …
```

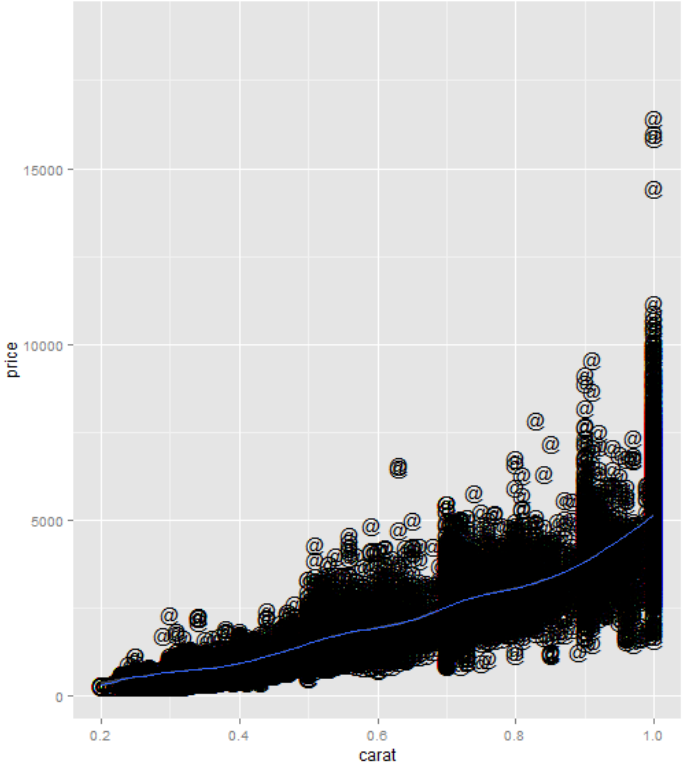
```
d <- ggplot(diamonds,
       aes(x=carat, y=price))
     + geom_polygon()
     + stat_smooth()
     + xlim(0.2, 1)
```



Calling ggplot

ggplot(*data, aesthetic mapping*)
(+ stat) (+ geom) (+ position) …

```
d <- ggplot(diamonds,
            aes(x=carat, y=price))
       + geom_text(label = '@')
       + stat_smooth()
       + xlim(0.2, 1)
```



How can you modify the default appearance of geoms and stats? Use parameters:

```
+ geom_smooth(method=lm)
+ stat_bin(binwidth = 100)
+ stat_summary(fun="mean_cl_boot")
+ geom_boxplot(outlier.colour = "red")
```

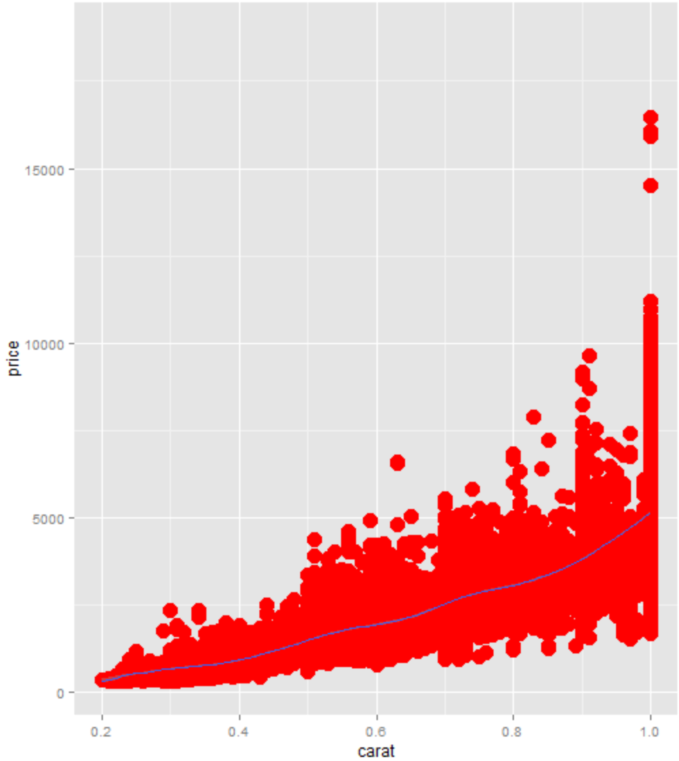
Any aesthetic can also be used as a parameter

```
+ geom_point(colour = "red", size = 5)
+ geom_line(linetype = 3)
```

Calling ggplot

```
ggplot(data, aesthetic mapping)
(+ stat) (+ geom) (+ position) …
```

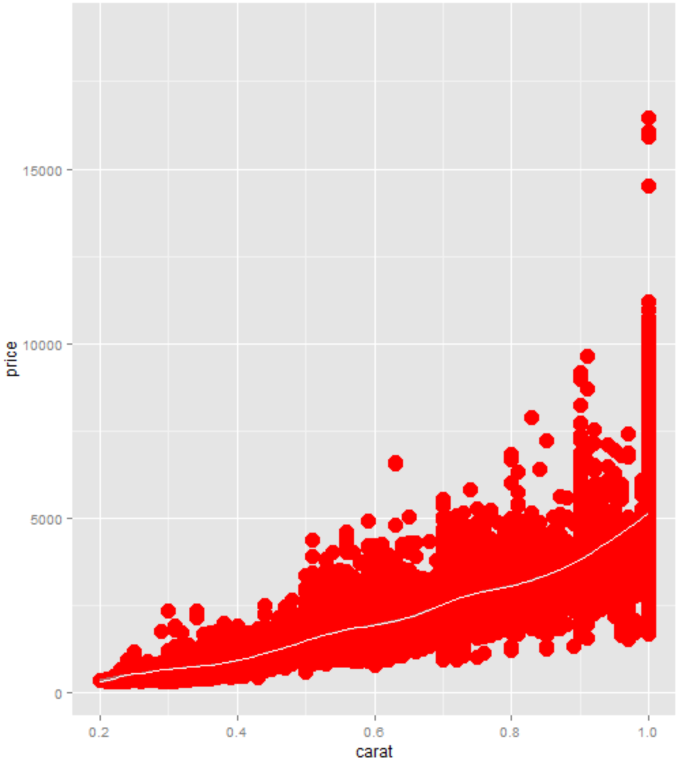
```
d <- ggplot(diamonds,
        aes(x=carat, y=price))
     + geom_point(size = 5,
          color = 'red')
     + stat_smooth()
     + xlim(0.2, 1)
```



Calling ggplot

```
ggplot(data, aesthetic mapping)
(+ stat) (+ geom) (+ position) …
```

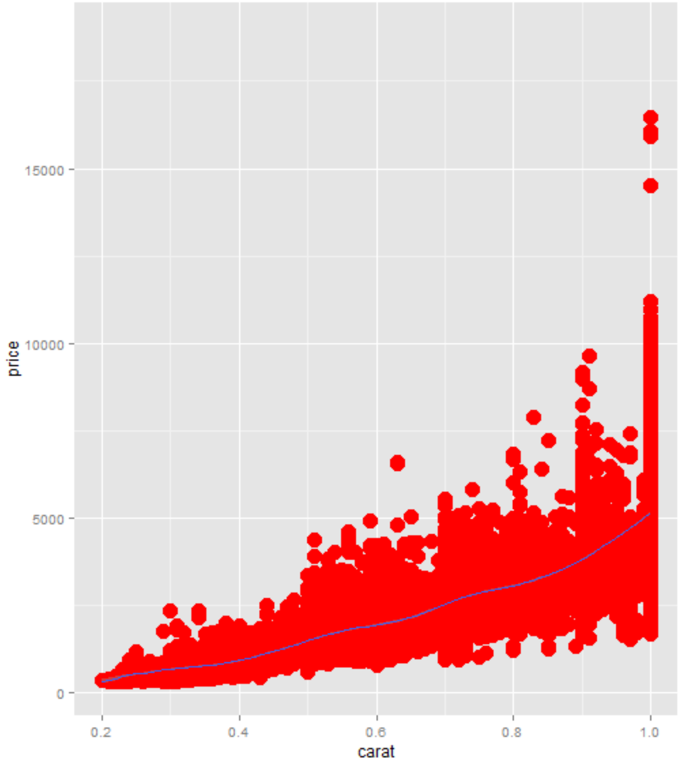
```
d <- ggplot(diamonds,
        aes(x=carat, y=price))
     + geom_point(size = 5,
          color = 'red')
     + stat_smooth()
     + geom_smooth(color ='white')
     + xlim(0.2, 1)
```



summary() works for plots!

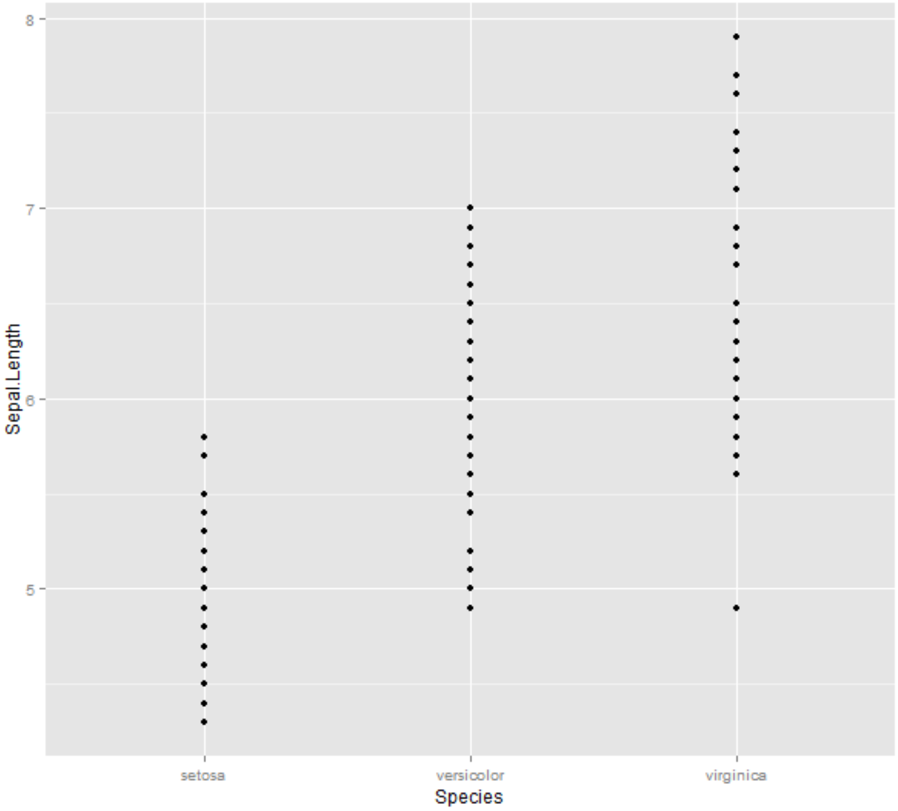
- Allows you see the aesthetic values for your plot or layers

```
> summary(d)
data: carat, cut, color, clarity, depth,
   table, price, x, y, z [53940x10]
mapping:  x = carat, y = price
scales:   x, xmin, xmax, xend,
   xintercept
faceting: facet_null()
-----------------------------------
geom_point: na.rm = FALSE, colour = red,
   size = 5
stat_identity:
position_identity: (width = NULL, height
   = NULL)
geom_smooth:
stat_smooth: method = auto, formula = y
   ~ x, se = TRUE, n = 80, fullrange =
   FALSE, level = 0.95, na.rm = FALSE
position_identity: (width = NULL, height
   = NULL)
```



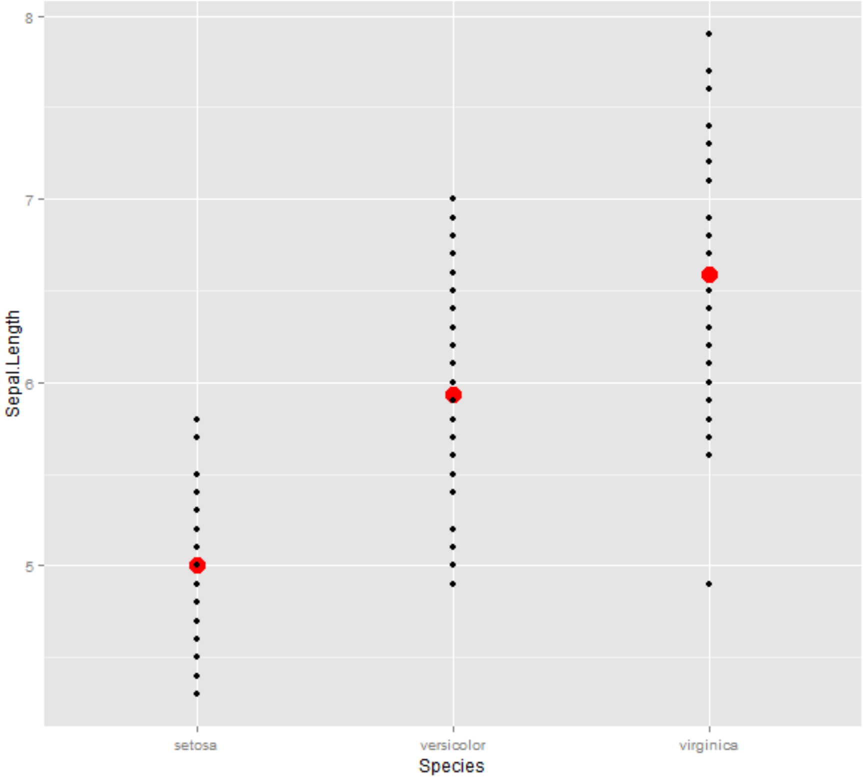
Recall the basic scatterplot

```
p <- ggplot(iris, aes(Species, Sepal.Length)) + geom_point()
```



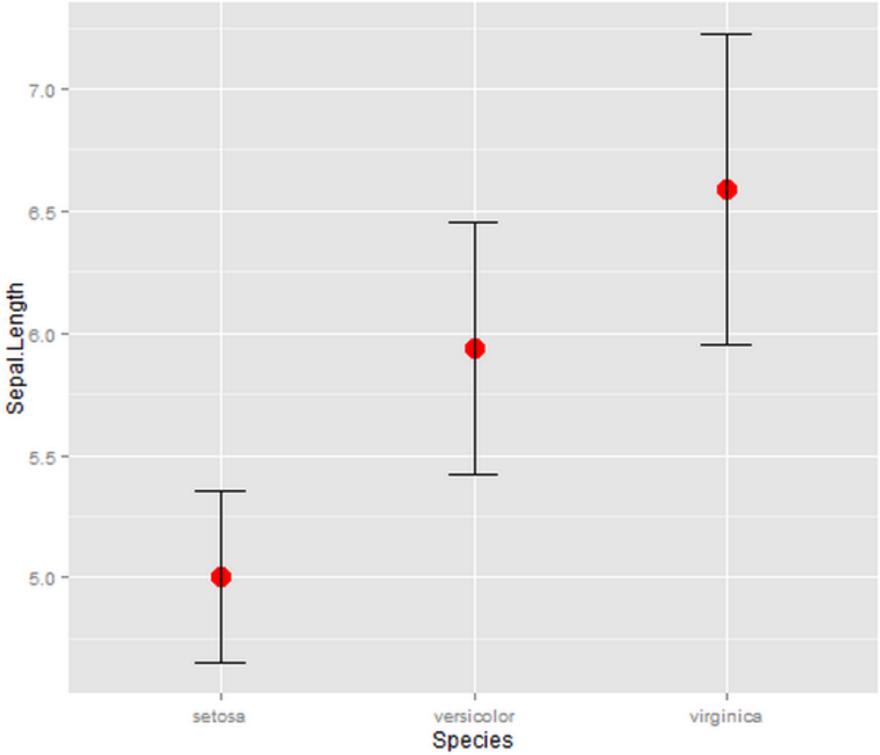
Statistics (*stat_*) functions apply statistical transformations that are used to summarize the data

```
p <- ggplot(iris, aes(Species, Sepal.Length)) +
geom_point()+ stat_summary(fun.y = mean, geom = "point",
color = "red", size = 5)
```



stat_summary is a popular choice for plotting means and variances

```
p <- ggplot(iris, aes(Species, Sepal.Length))
+ stat_summary(fun.y = mean, geom = "point", color = "red", size = 5)
+ stat_summary(fun.data = "myFunc", geom = "errorbar", width = 0.2)
```



```
myFunc = function(x) {
result = c(mean(x) - sd(x),
mean(x) + sd(x))
names(result) = c("ymin", "ymax")
result
}
```

See Wickham Chapter 4, Table 4.4 for a complete stat list

Special internal variables

- Some stat_ functions produce new internal variables from data, dynamically
- stat_bin produces *count* and *density*
- If you want to map an aesthetic to one of

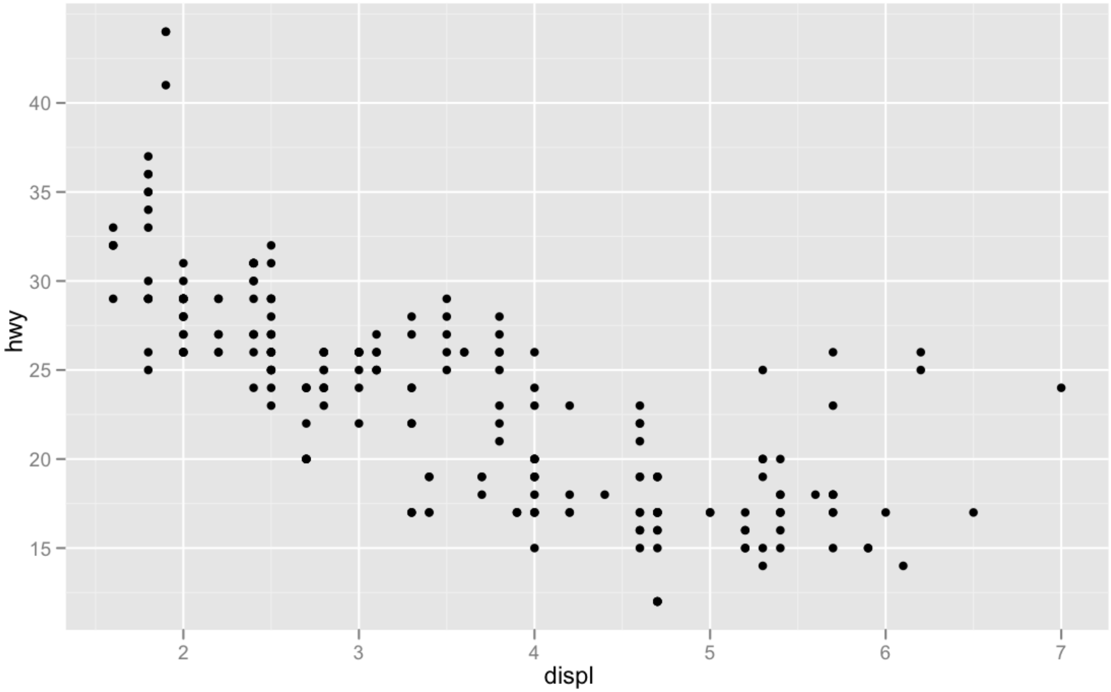
these new variables, surround it with double periods, like this: `..foo..`

```
ggplot(diamonds, aes(x=price))
+ geom_histogram(aes(y = ..density..))
+ geom_histogram(aes(colour = ..count..))
```

Smoothing

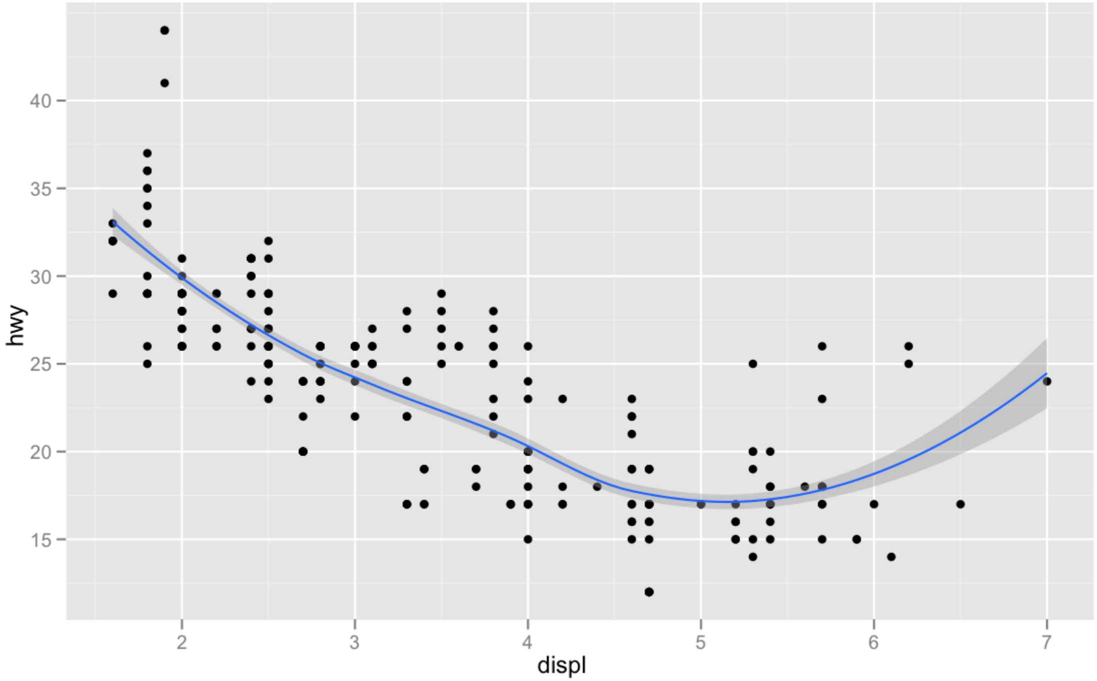
Modeling trends in data with stat_smooth()

```
p <- ggplot(mpg, aes(displ, hwy))
+ geom_point()
```



Modeling trends in data with stat_smoothing

```
p <- ggplot(mpg, aes(displ, hwy))
+ geom_point() + stat_smooth()
```



Dataset: 1970 U.S. Government draft lottery for birthdates

- In an attempt to expose male youth fairly to the risk of being drafted, a lottery was held to allocate birthdates at random.
- Applied to eligible men aged 19 to 26 prior to January 1, 1970, and so included births taking place in some leap years.
- 366 capsules, each containing a unique day of the year, were successively drawn from a container.
  - The first date drawn (September 14) was assigned rank 1
  - The second date drawn (April 24) was assigned rank 2
  - Those eligible for the draft who were born on September 14 were called first for physicals, then those born on April 24 were tapped, and so on.

Source: http://www.amstat.org/publications/jse/v5n2/datasets.starr.html

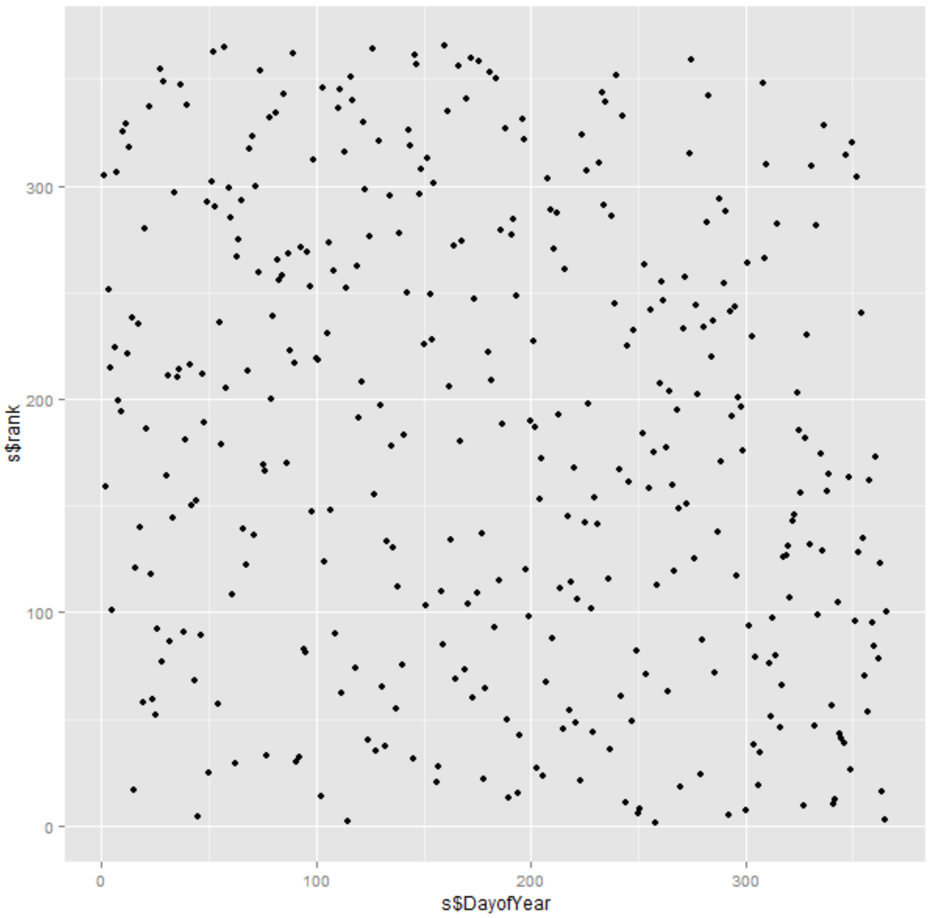
Lottery data format

Columns

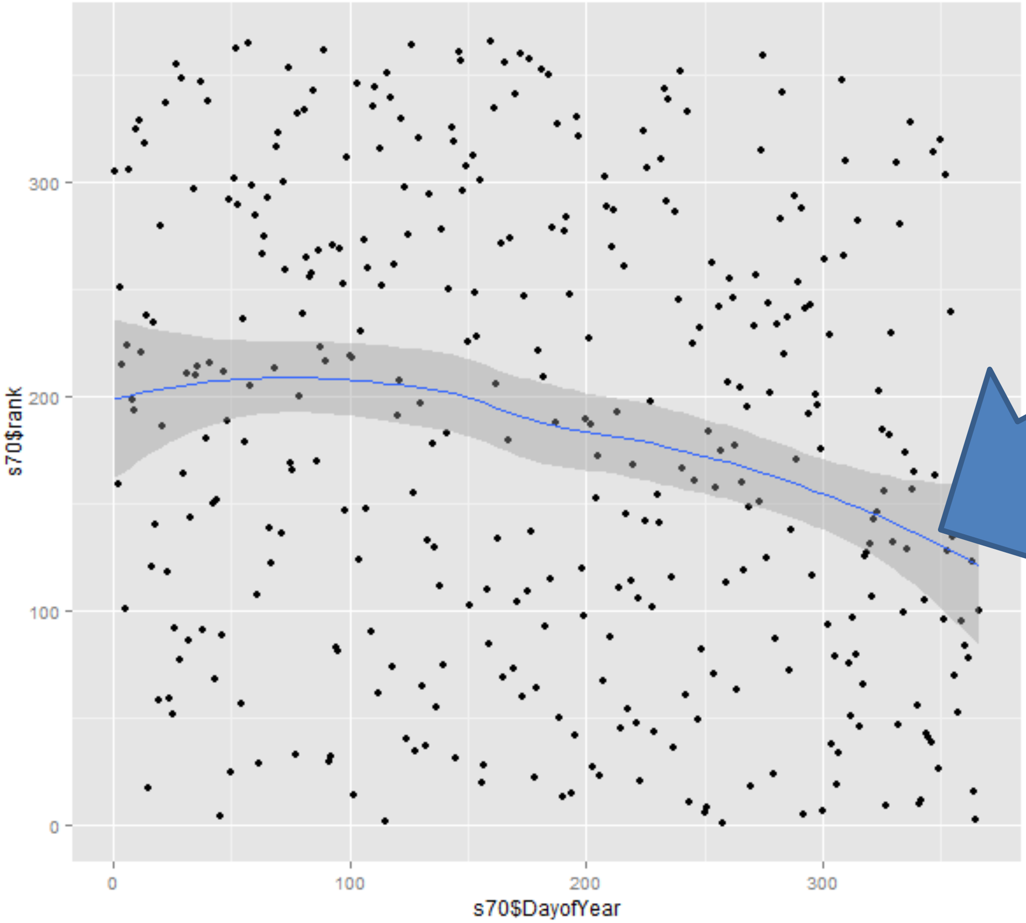
- 1 Day of the year from 1 to 366
- 2 Rank assigned to day
- 3 Month of the year between 1 and 12

```
s <- read.table("draft70yr.dat.txt",
     col.names=c("DayofYear", "Rank", "Month"))
```

The lottery-based rank of each day in 1970: A random scatterplot?



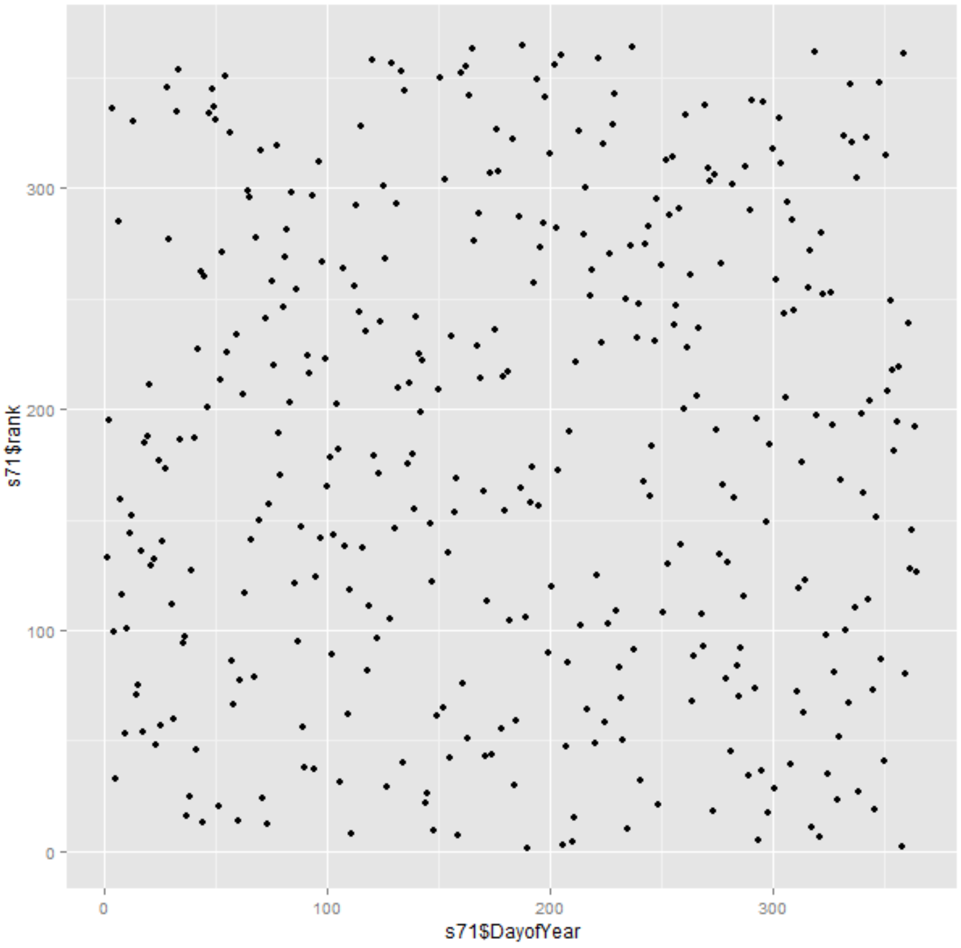
Let's try: 1970 + stat_smooth()



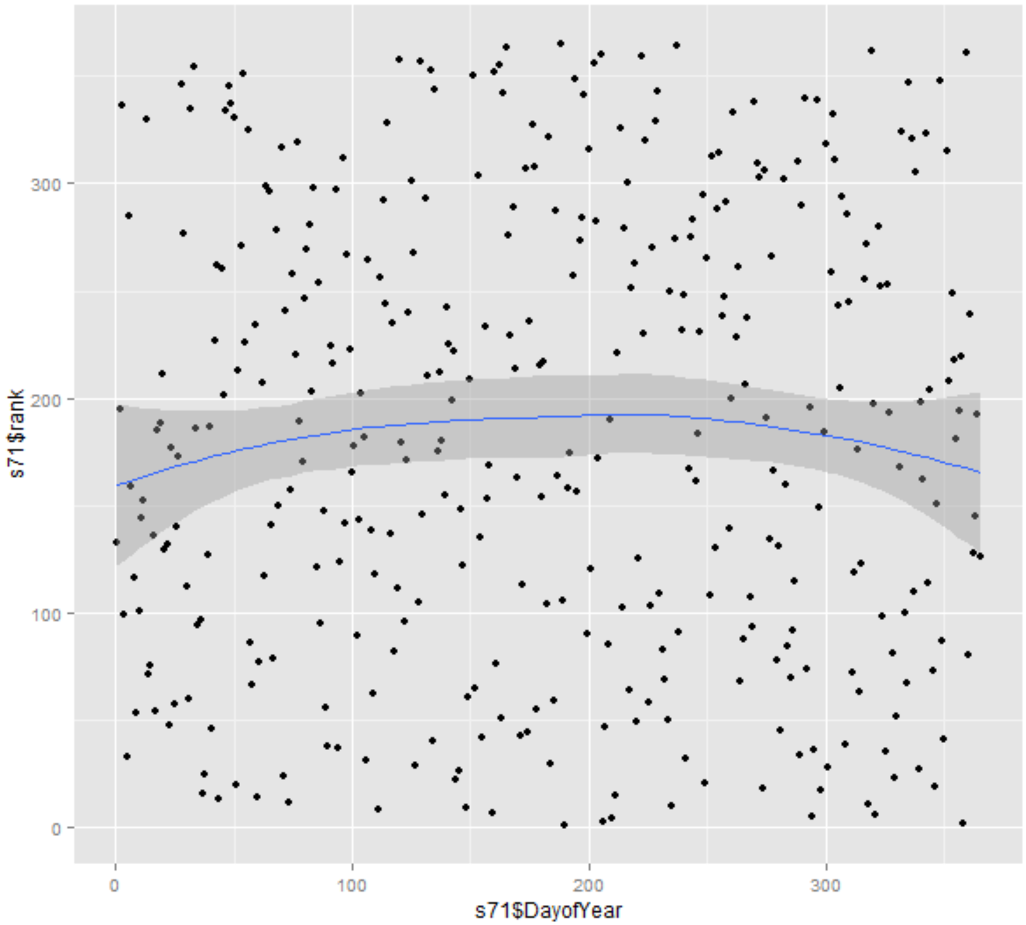
Statistically significant bias!

© 2016 Chris Teplovs

This is the 1971 lottery: A random scatterplot?



1971 + stat_smooth()

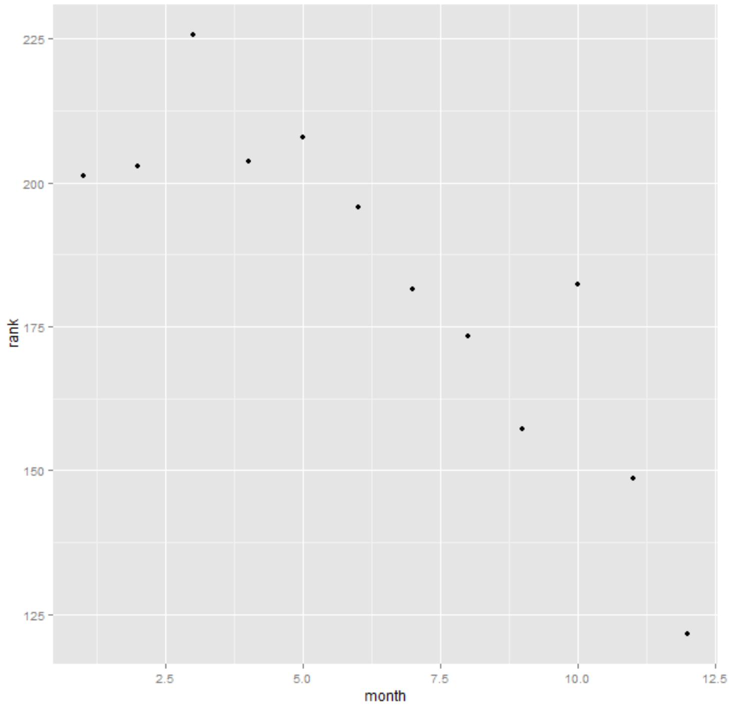


stat_summary month-by-month

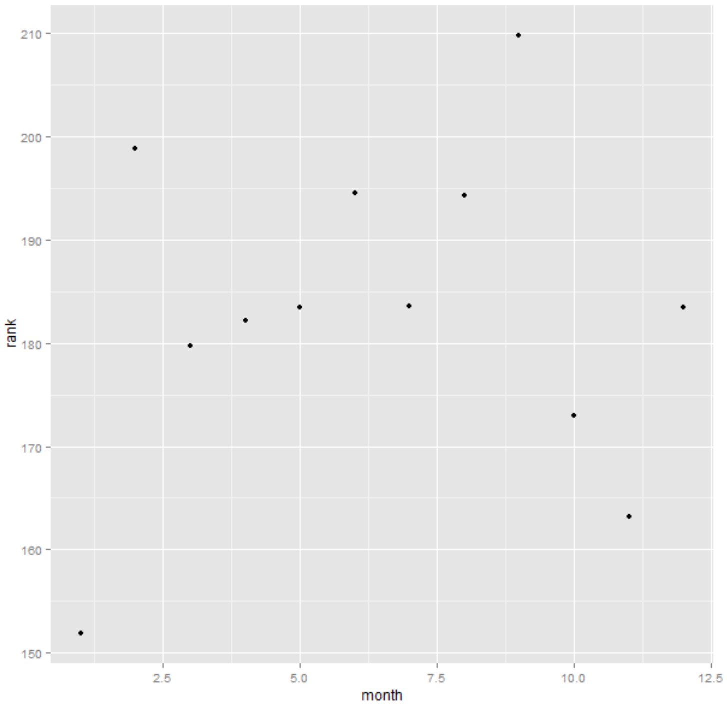
Source: http://www.amstat.org/publications/jse/v5n2/datasets.starr.html

```
ggplot(s70, aes(month, rank)) + stat_summary(fun.y=mean, geom="point")
```

"The capsules were put in a box month by month, January through December, and subsequent mixing efforts were insufficient to overcome this sequencing."



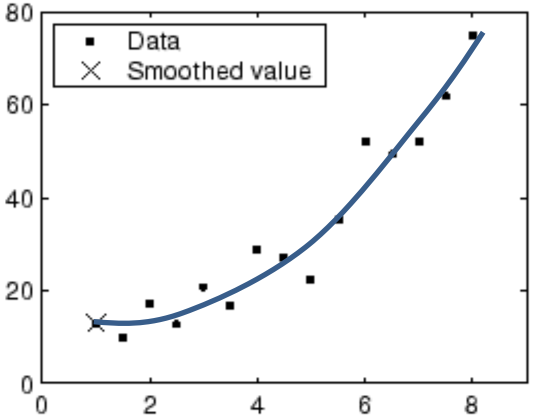
1970 lottery



1971 lottery

How smoothing works

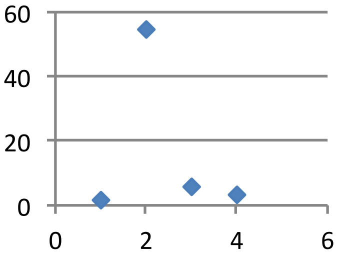
- Typically compute a moving average or regression in a local neighborhood
- Commonly-used methods:
  - Median: Replace each data point with the median of the *k* neighbor values
  - LOWESS: LOcally WEighted Scatterplot Smooth



Source: http://www.mathworks.com/help/curvefit/smoothing-data.html

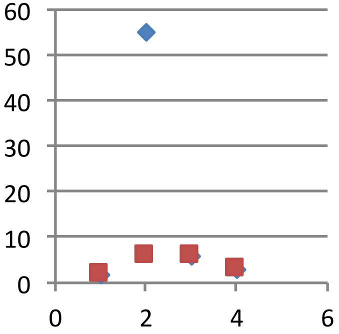
Median smoothing

- Problem: Lots of noisy variation in the data
- Idea: Reduce noise by replacing each data entry with the *median* over a window of neighboring entries
- Set of neighbors is called the window
  - At boundaries, e.g. no entry preceding first value: just repeat it.



Input:

```
x = [2 55 6 3]
```



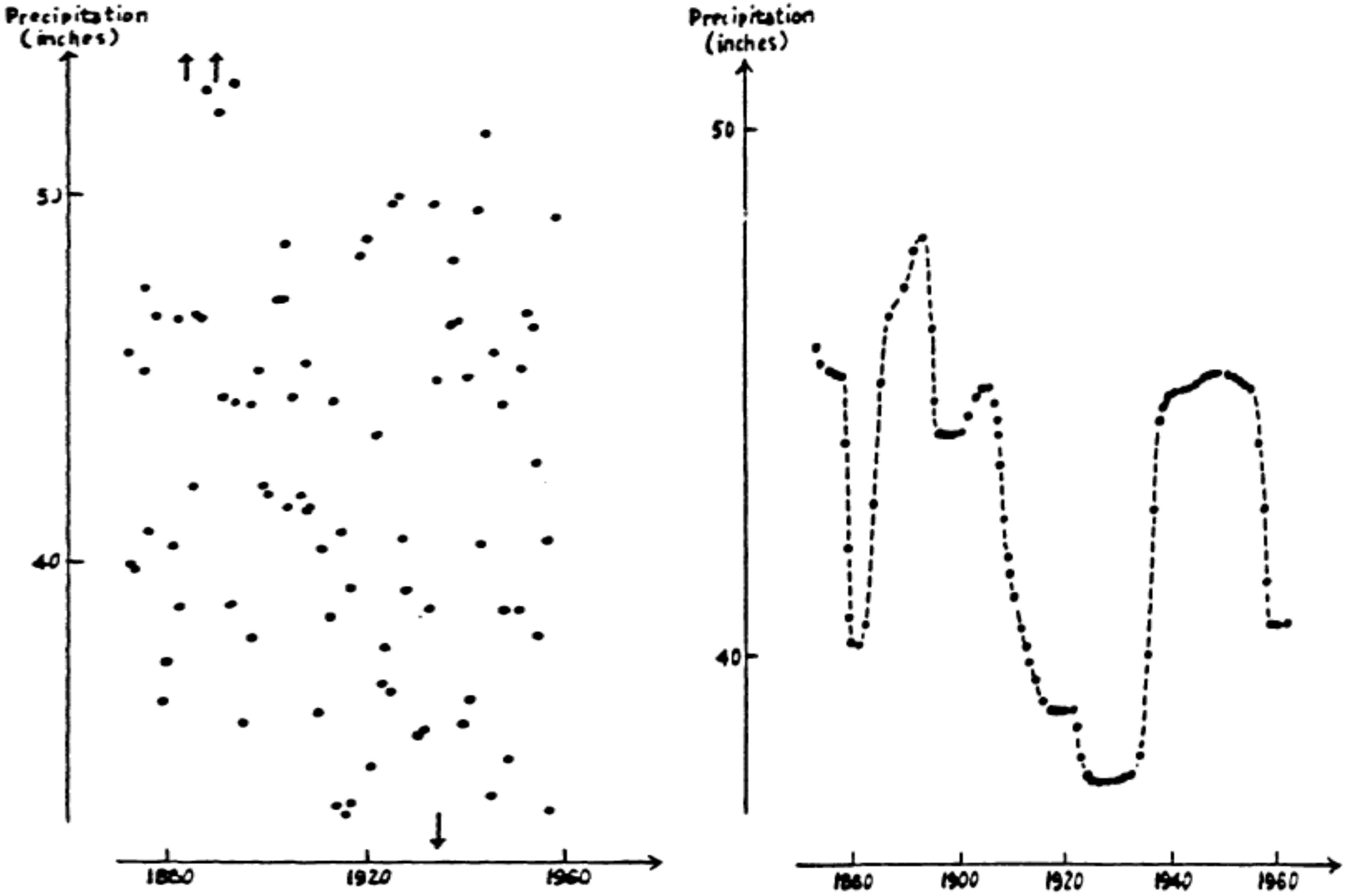
Using window size = 3, median filtered output signal y will be:

```
y[1] = Median[2 2 55] = 2
y[2] = Median[2 55 6] = Median[2 6 55] = 6
y[3] = Median[55 6 3] = Median[3 6 55] = 6
y[4] = Median[6 3 3] = Median[3 3 6] = 3
```

Output:

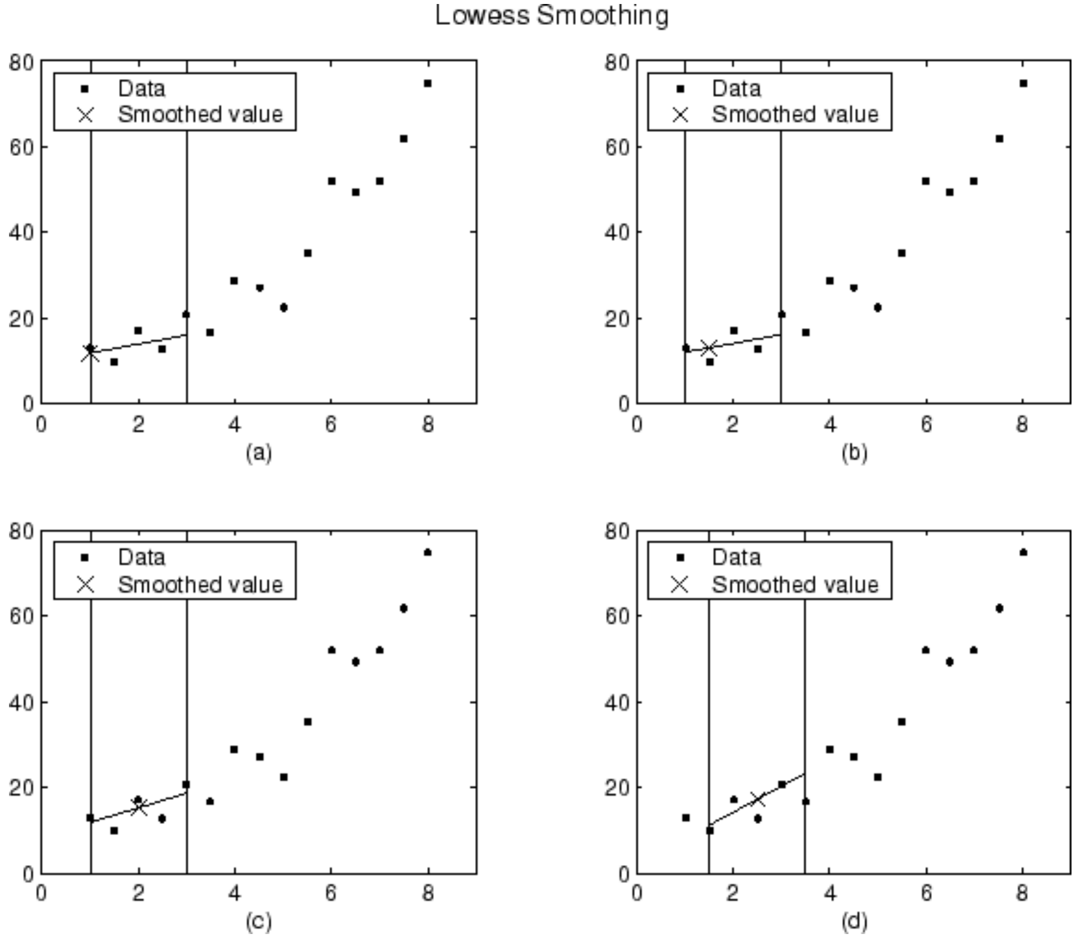
```
y = [2 6 6 3]
```

Median smoothing at work: New York City rainfall over 100 years

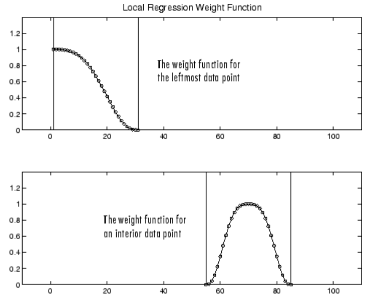


Source: J. Tukey, Exploratory Data Analysis

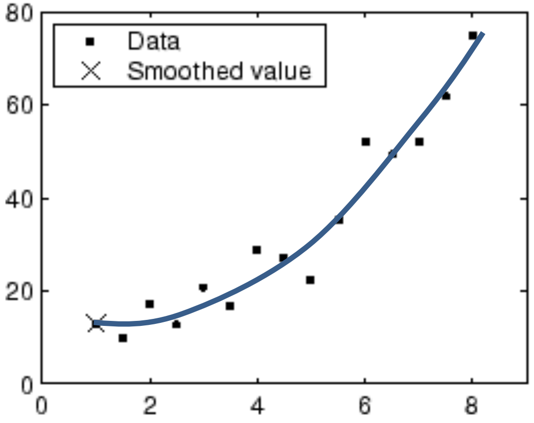
LOWESS smoothing gives more adjustable, less 'spiky' fits compared to median smoothing



How much do neighbors influence this prediction?
Specified using a *kernel function*:



Final result:

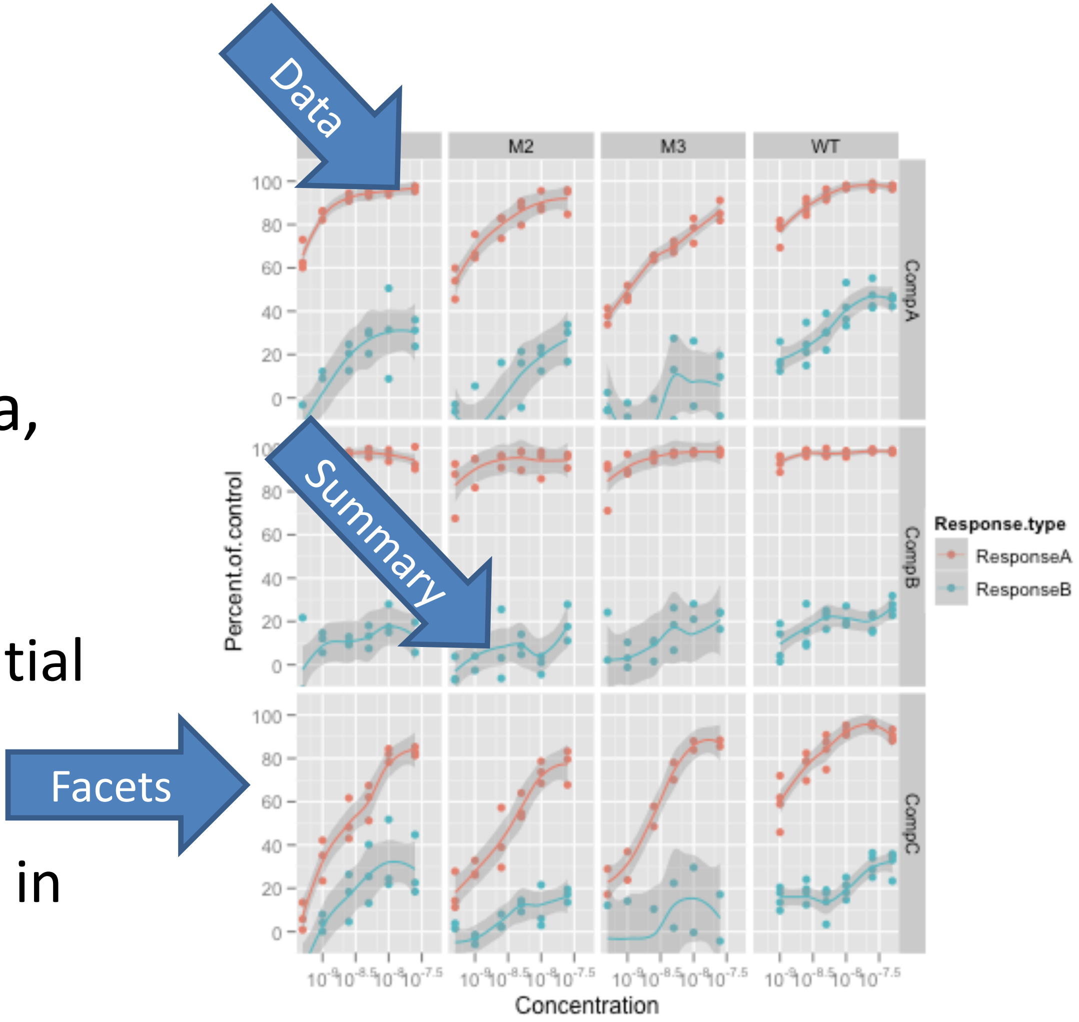


© 2016 Chris Teplovs

Adding layers

Layering strategy. Why add a layer?

1. Display data
2. Display statistical summary of the data
3. Additional metadata, context, annotations
   - Facets
   - e.g. map as geospatial background layer
   - Highlight or label important features in the data



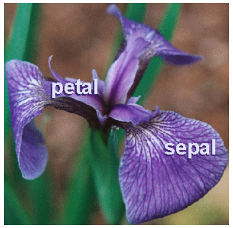
Data

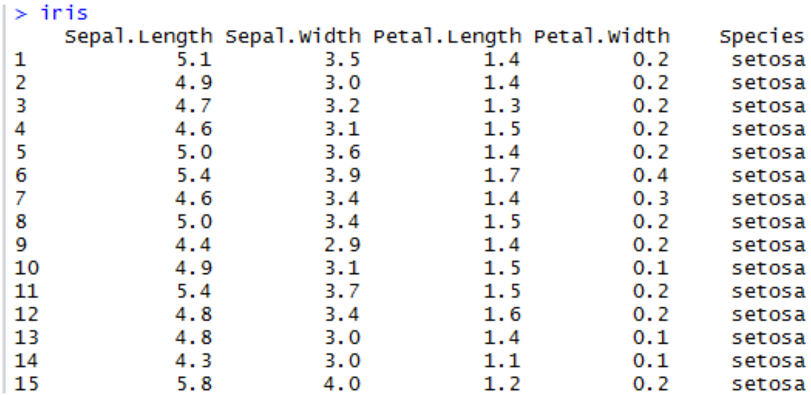
Summary

Facets

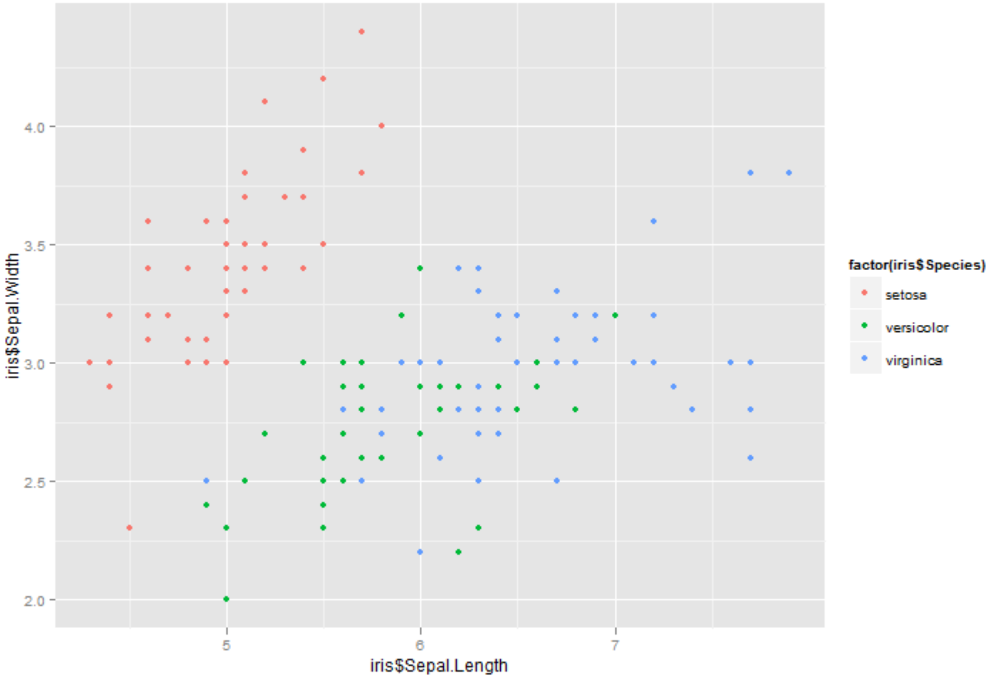
© 2016 Chris Teplovs

The Iris dataset: Fisher (1936)





```
> iris
   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1           5.1         3.5          1.4         0.2     setosa
2           4.9         3.0          1.4         0.2     setosa
3           4.7         3.2          1.3         0.2     setosa
4           4.6         3.1          1.5         0.2     setosa
5           5.0         3.6          1.4         0.2     setosa
6           5.4         3.9          1.7         0.4     setosa
7           4.6         3.4          1.4         0.3     setosa
8           5.0         3.4          1.5         0.2     setosa
9           4.4         2.9          1.4         0.2     setosa
10          4.9         3.1          1.5         0.1     setosa
11          5.4         3.7          1.5         0.2     setosa
12          4.8         3.4          1.6         0.2     setosa
13          4.8         3.0          1.4         0.1     setosa
14          4.3         3.0          1.1         0.1     setosa
15          5.8         4.0          1.2         0.2     setosa
```



```
qplot(iris$Sepal.Length, iris$Sepal.Width, colour=factor(iris$Species))
```

Construct the plot

- ggplot takes two arguments (yes there are more, but this is a slimmed down version)
  - data
  - aes

```
> p = ggplot(iris, aes(Sepal.Length, Sepal.Width,
  colour=factor(Species)))
```

- If we run this:

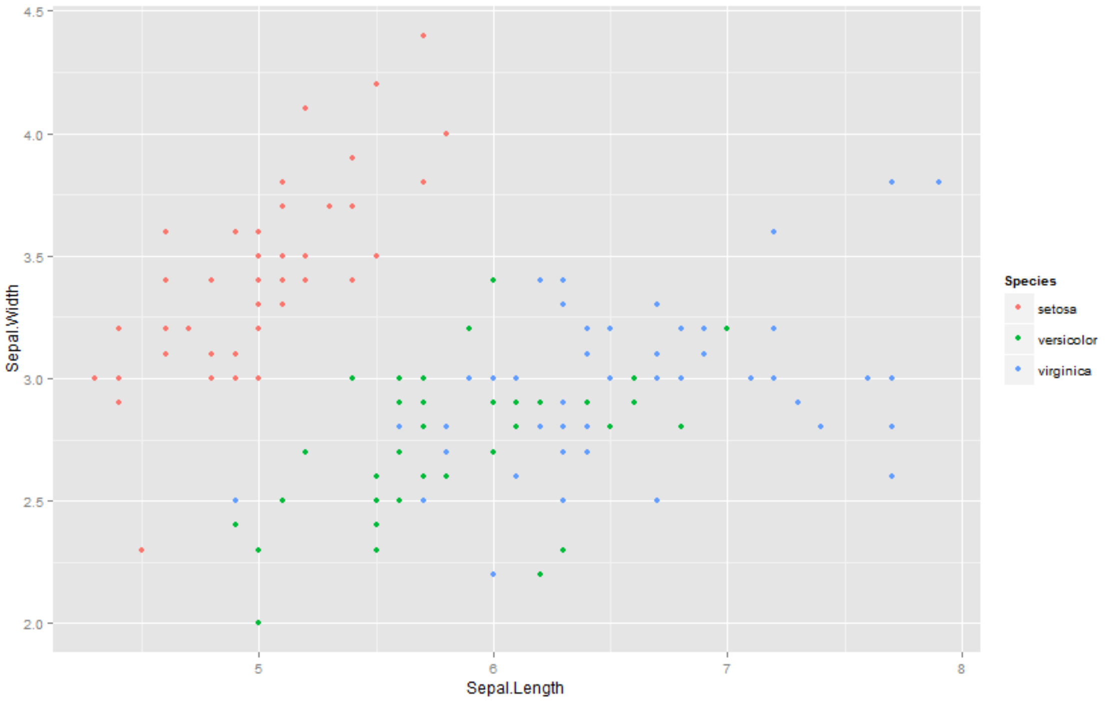
```
> p
Error: No layers in plot
```

Adding the first layer

- Now this data needs to be added to a layer to visualize

```
p + layer(geom = "point")
```

- Result:

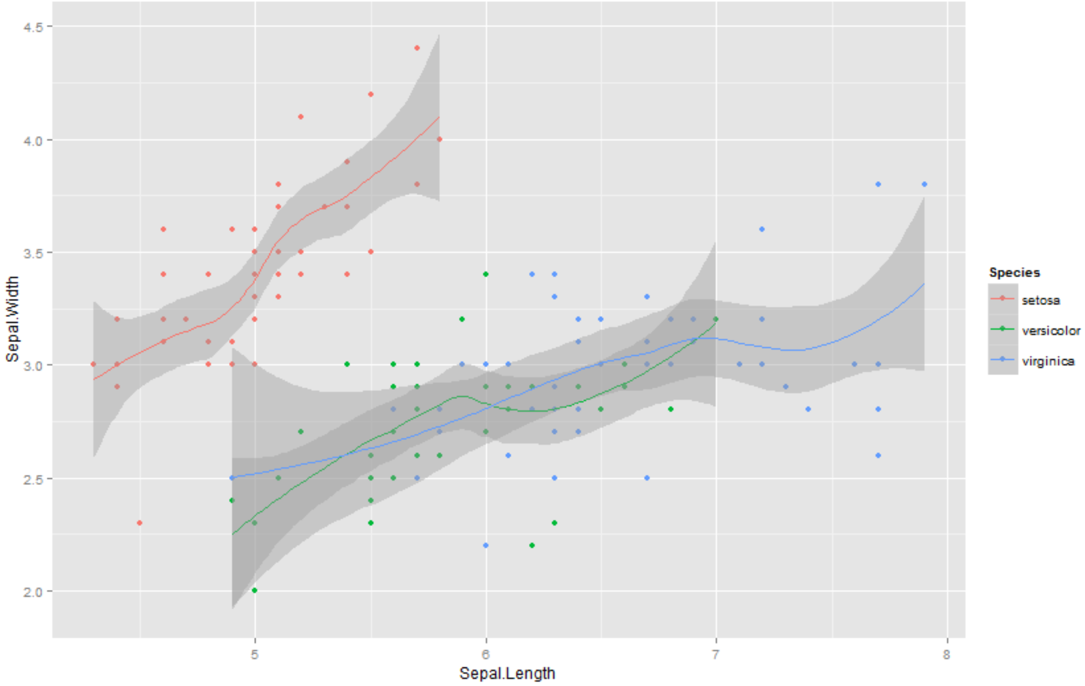


Adding another layer

- Now this data needs to be added to a layer to visualize

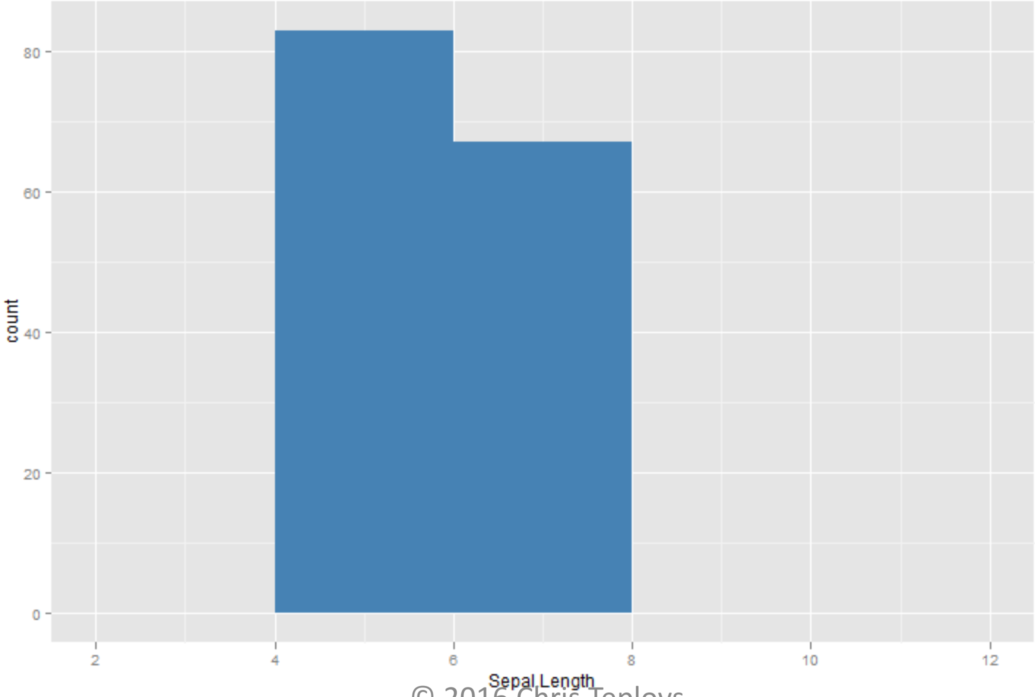
```
p + layer(geom = "point")+layer(stat="smooth")
```

- Result:



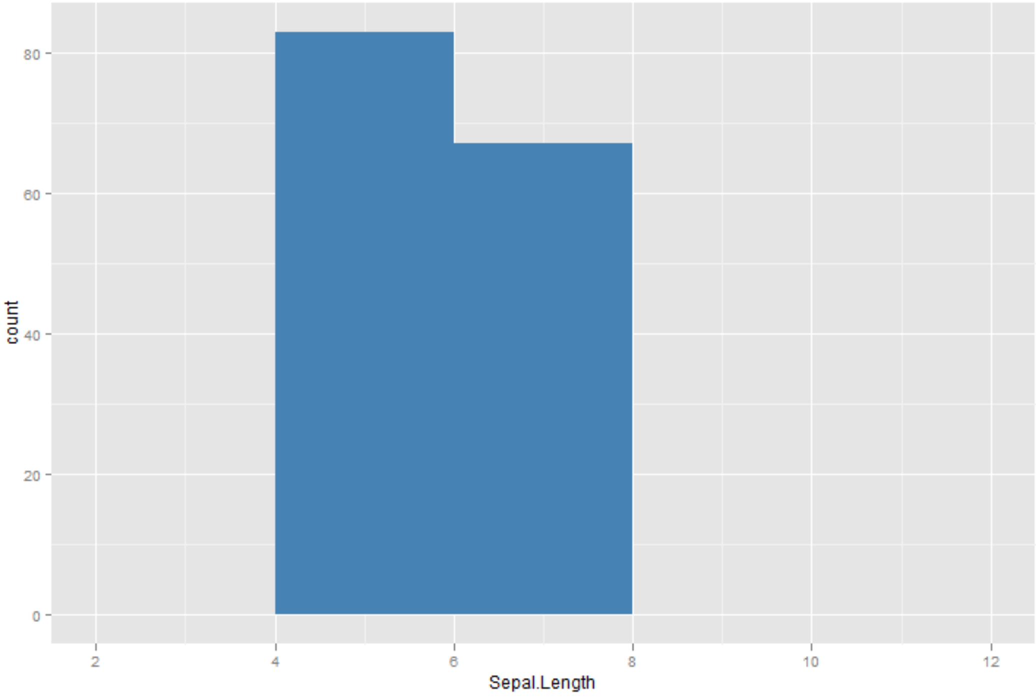
Plotting using layers()

```
> p = ggplot(iris, aes(x = Sepal.Length))
> p + layer(geom = "bar",
            geom_params = list(fill = "steelblue"),
            stat = "bin", stat_params = list(binwidth = 2))
```



Plotting using geom shortcut with params

```
> p = ggplot(iris, aes(x = Sepal.Length))
> p + geom_histogram(binwidth = 2, fill =
  "steelblue")
```



More advanced layers

```
ggplot(data, mapping) +
layer(stat = "", geom = "", position = "", geom_params = list(), stat_params = list())
```

The _params functions can encapsulate lists

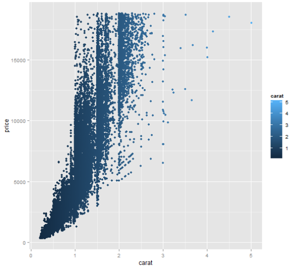
- `geom_params = list(color = "red", alpha = 0.5)`
- `stat_params = list(method = "lm", se = F)`

Usually won't write out the full specification of a layer, but use a shortcut:

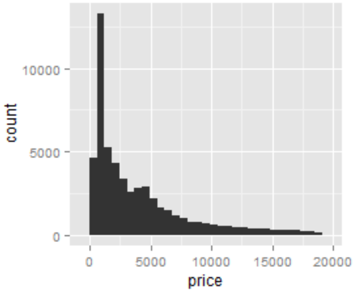
- `geom_smooth()`
- `stat_summary()`
- `...`

- Every geom has a default statistic, every statistic a default geom (but can override)

```
d <- ggplot(diamonds, aes(x=carat, y=price))
d + geom_point(aes(colour = carat))
  + scale_colour_brewer()
```



```
ggplot(diamonds) + geom_histogram(aes(x=price))
```



Annotating a plot with title and axis labels

- Adding a title to your plot

```
    + ggtitle( "The plot title")
```

- Adding x-axis label

```
    + xlab("Year")
```

- Adding y-axis label

```
    + ylab("Mean combined MPG")
```

Axes and legends are called 'guides'

- Properties are set using the themes() and guides() functions
- Most legend properties are computed automatically by ggplot
  - But some adjustments are possible
- Examples:
  - Setting vertical x-axis text. Add this:

```
+ theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

  - Setting number of columns in the legend. Add this:

```
+ guides(col = guide_legend(ncol = 2))
```

See ggplot2 Wickham, Section 8.1 for more on themes.

For more on guide_legend() see:
http://cloud.github.com/downloads/hadley/ggplot2/guide-col.pdf

SQL

Connecting to SQL databases

- Some datasets exist only in relational databases.
- Data may be dynamic and require repeated analysis over time.
- Relational databases can make working with huge datasets easier.
- Two principal ways to connect with databases in R:
  - Open DataBase Connectivity (ODBC)
    - Generic standard, available on many computers and DB types, but can be slow.
  - DBI package + specialized driver package
    - Support for only specific database types, but often better performance.
    - We'll use this one, with SQLite

```
> install.packages("DBI")
> install.packages("RSQLite")
```

Required

Database retrieval in R using SQLite

- Assume we've created a sqlite2 (or 3) database using Python
  - Example code: create_database.py
- Then, we retrieve data from the database in R

```
library(DBI)
library(RSQLite)
library(ggplot2)
dbdriver = dbDriver("SQLite")
connect  = dbConnect(dbdriver, dbname = "vehicles.db")
vehicles = dbGetQuery(connect, "select * from vehicle")
head(vehicles)
```

- Example code: ConnectingToDatabase.R
- For other databases, check out http://www.statmethods.net/input/dbinterface.html

What you should know

- What R factors and levels are, and how to use them in exploratory data analysis
- ggplot analysis
  - aesthetics
  - geoms
  - stats
  - guides
- How to fetch data frames by running SQL queries against a SQLite DB in R

Next up: Start Homework #3

- Part 1: use Python to create a local SQLite DB
  - See create_database.py and cars database in examples-week3.zip in Week 3 resources
- Part 2: use R and ggplot2 to compute aggregated statistics and plots

Supplemental slides

Which data type is most appropriate for each of the following variables?

Medical experiment dataset

Subject_ID
Name
Treatment
Gender
Number of siblings
Address
Race
Eye color
Birth city
Birth state

Data frame summary with str()

```
> str(diamonds)
'data.frame':         53940 obs. of  10 variables:
 $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
 $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
 $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
 $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
 $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
 $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
 $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
 $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
 $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
 $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...

> str(mtcars)
'data.frame':         32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Groups: controlling which rows go with which graphical element

- Without grouping

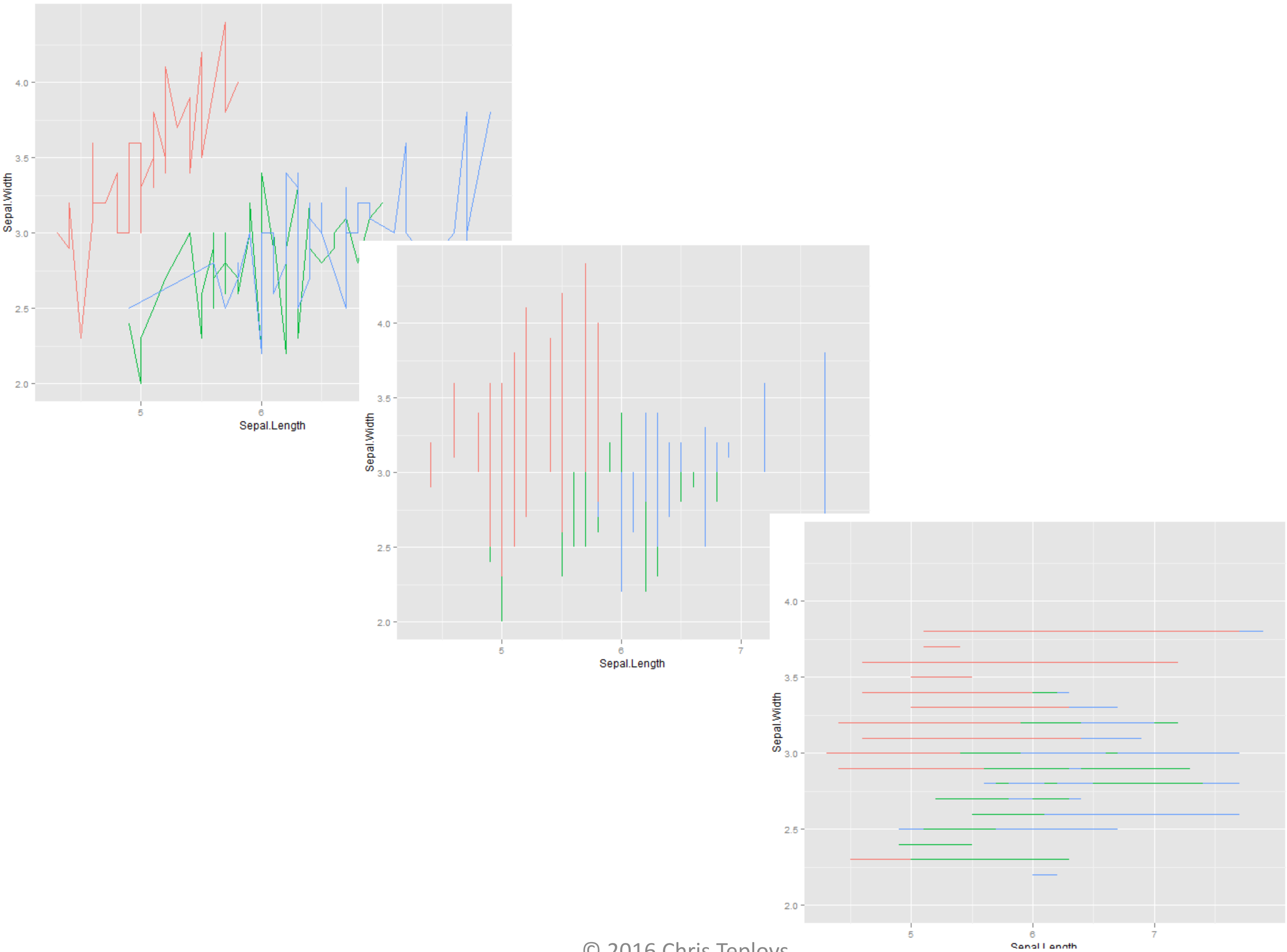
```
> ggplot(iris, aes(Sepal.Length, Sepal.Width, color =
  factor(Species))) + geom_line()
```

- Grouping on length

```
> ggplot(iris, aes(Sepal.Length, Sepal.Width,
  group = Sepal.Length, color = factor(Species))) +
  geom_line()
```

- Grouping on width

```
> ggplot(iris, aes(Sepal.Length, Sepal.Width,
  group = Sepal.Width, color = factor(Species))) +
  geom_line()
```


4.0
3.5
3.0
2.5
2.0
Sepal.Width
5
6
7
Sepal.Length

Converting factors back

- Add a cylinders factor to mtcars

```
> mtnew <- data.frame(mtcars, factor(mtcars$cyl, labels = c("4", "6", "8")))
> str(mtnew)
'data.frame':           32 obs. of  12 variables:
 $ mpg                                 : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl                                 : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp                                : num  160 160 108 258 360 ...
 [other variables]
 $ factor.mtcars.cyl..labels...c..4....6....8...: Factor w/ 3 levels "4","6","8": 2 2 1 2 3 2 3 1 1 2 ...
> mean(mtcars$factor.mtcars.cyl)
[1] NA
Warning message:
In mean.default(mtcars$factor.mtcars.cyl) : argument is not numeric or logical: returning NA
```

```
> levels(mtnew$factor.mtcars.cyl)[mtnew$factor.mtcars.cyl]
 [1] "6" "6" "4" "6" "8" "6" "8" "4" "4" "6" "6" "8" "8" "8" "8" "8"
"8" "4" "4" "4" "4" "8" "8" "8" "8" "4" "4" "4" "8" "6" "8" "4"
> as.numeric(levels(mtnew$factor.mtcars.cyl)[mtnew$factor.mtcars.cyl])
[1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
> as.numeric(as.character(mtnew$factor.mtcars.cyl))
[1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```