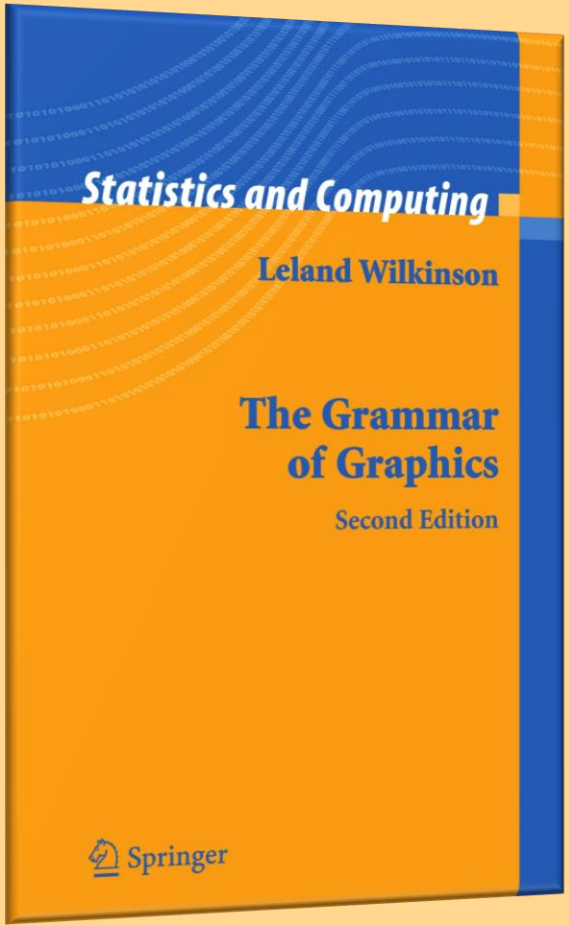
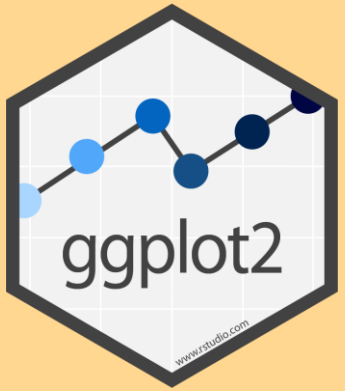
Visualising data in R

Graduate skills class OU22

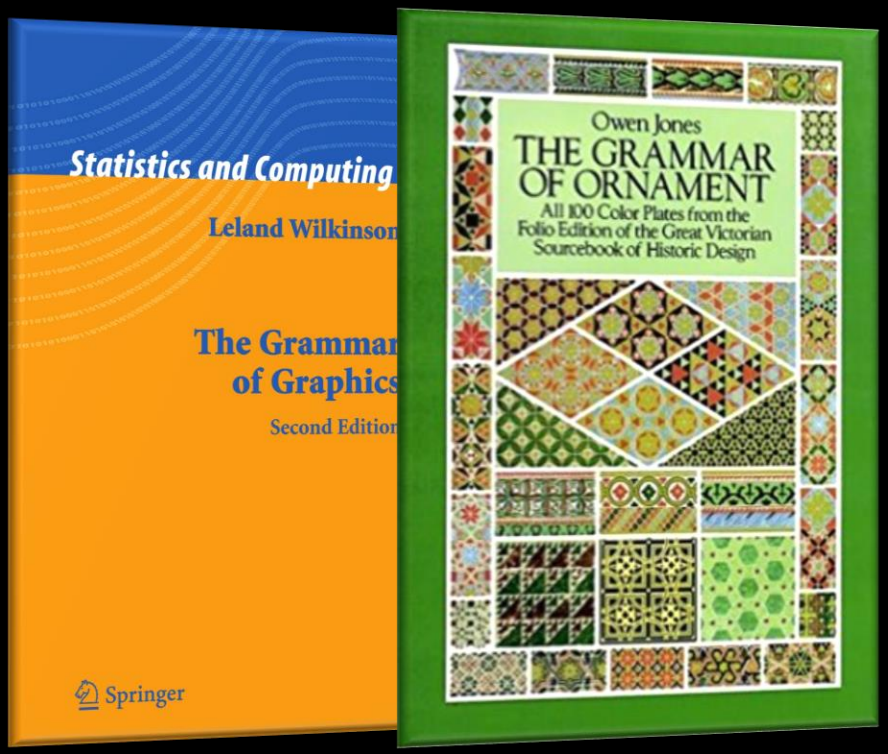
Practical:
Monday 19 February, 2-3pm, SW01.
OR Wednesday 21 February, 3-4pm, SW01.





The *Grammar of Graphics* codifies some standard patterns in plotting data. It doesn't try to be exhaustive, nor is it a guide to good style.

ggplot2 will simplify your life — if you learn the way it thinks, and if you don't step outside its scope.



Department of Compute ×

www.cl.cam.ac.uk/~djw1005/Teach/ggvis/

UNIVERSITY OF CAMBRIDGE

Search

Contact us | A–Z | Advanced search

Department of Computer Science and Technology

Computer Laboratory › Teaching › Courses 2017–18 › OU22: visualizing data with R

Course pages 2017–18

OU22: visualizing data with R

Course materials

Lecturer: Dr Damon Wischik
Taken by: Part III, ACS, PhD
Course admin: [Moodle]

No. of lectures and practical classes: 1 + 1

Aims

The R language is widely used by data scientists. It is concise and expressive for handling data, and it has a powerful graphics library called ggplot. This course will show you how to get data into R, how to do simple data manipulation, and how to use ggplot for flexible and versatile data visualization. See examples at the R graph gallery.

Resources

- Invoking R+ggplot from Jupyter/Python [notebook]
- Running R+ggplot in Jupyter/Python [notebook]
- Slides (to appear)
- Practical (to appear)

Installation instructions (Ubuntu, Windows)

On Windows, first follow Microsoft's instructions to install the Windows Subsystem for Linux, with Ubuntu 16.04.

Installing Jupyter and Python.
Install git, Python, and Jupyter. Ubuntu 16.04 uses Python 3.5 for many of its system scripts, so it's not a good idea to upgrade Python systemwide, so it's best to install Jupyter on top of Python 3.5:

```
sudo apt install python3-pip python3-dev git
sudo pip3 install --upgrade pip
sudo pip3 install jupyter
```

Install Python 3.6 and make it available as a programming environment within Jupyter:

```
sudo add-apt-repository ppa:jonathonf/python-3.6
```

Setting up Jupyter and R

Python notebook + R plots

First, follow the installation instructions for Jupyter, Python, R, and R/Jupyter.

Warning. This doesn't work from Azure notebooks.

In [ ]:
```
%%capture
# Jupyter is very chatty. If you start a cell with %%capture, it suppresses all output
# (including any helpful warning or error messages).

# Load the rpy2 extension for Jupyter.
# (There will now be an R kernel sitting alongside the Python kernel.)
%load_ext rpy2.ipython

# You can run a one-line R command with %R. Here, I'm loading a plotting library.
# It will be available for the rest of this Jupyter session.
%R library(ggplot2)
%R library(ggthemes)
```

In [5]:
```
import urllib
import pandas
import numpy as np

# Load in a dataset
iris = pandas.read_csv(urllib.request.urlopen('https://teachingfiles.blob.core.windows.net/founds/iris.csv'))

# Have a quick look
iris.iloc[np.random.choice(len(iris), 3, replace=False)]
```

Out[5]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
53	5.5	2.3	4.0	1.3	versicolor
67	5.8	2.7	4.1	1.0	versicolor
44	5.1	3.8	1.9	0.4	setosa

We can inject a Python object into R using `-i`. It will be copied across, and available in R with the same name.

In [12]:
```
%R -i iris
```

You can't copy arbitrary variables across from R to Python. The `rpy2` package knows about some standard simple datatypes (including pandas `DataFrames`), and it is able to translate them, but if you copy across something weird it will fail.

```
class X: pass
x = X()
```

Invoking ggplot2 commands from a Python/Jupyter notebook

jupyter ggplot Last Checkpoint: 22 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted R

Visualizing data with R

In [2]:
```
# Load in some packages for plotting and data manipulation.
library(ggplot2)
library(ggthemes)
library(data.table)
```

In [4]:
```
# Set the plotting size (in inches)
options(repr.plot.width=4, repr.plot.height=2.5)
```

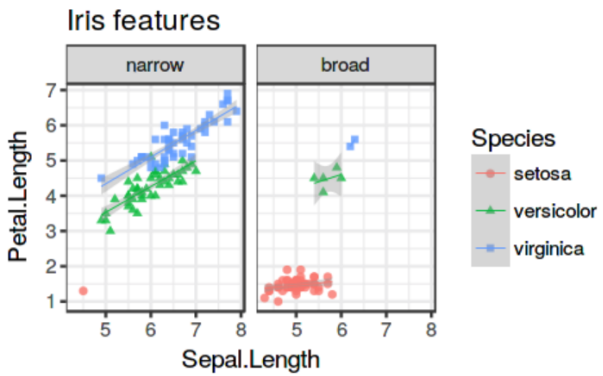
In [5]:
```
# Fetch a dataset in CSV format
iris <- fread('https://teachingfiles.blob.core.windows.net/founds/iris.csv')
# Print out some sample rows
iris[sample(nrow(iris),4)]
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.0	3.4	1.6	0.4	setosa
6.5	3.0	5.5	1.8	virginica
5.0	3.5	1.3	0.3	setosa
6.7	2.5	5.8	1.8	virginica

In [12]:
```
iris[, shape := cut(Sepal.Width/(Sepal.Width+Sepal.Length), 2, labels=c('narrow','broad'))]

ggplot(data=iris) +
    geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species), size=1.3, alpha=.8) +
    stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species), method='lm', size=.2) +
    facet_wrap(~shape) +
    scale_y_continuous(breaks=seq(0,10)) +
    ggtitle('Iris features') +
    theme_bw()
```

```
Warning message in qt((1 - level)/2, df):
"NaNs produced"
```



Running ggplot2 from an R/Jupyter notebook

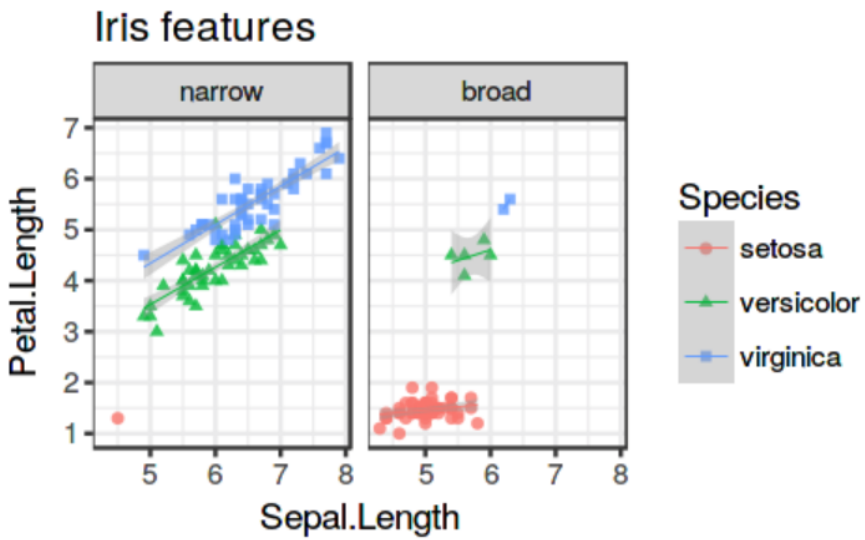
```
# Fetch a dataset in CSV format
iris <- fread('https://teachingfiles.blob.core.windows.net/founds/iris.csv')
# Print out some sample rows
iris[sample(nrow(iris),4)]
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.0	3.4	1.6	0.4	setosa
6.5	3.0	5.5	1.8	virginica
5.0	3.5	1.3	0.3	setosa
6.7	2.5	5.8	1.8	virginica

```
iris[, shape := cut(Sepal.Width/(Sepal.Width+Sepal.Length), 2, labels=c('narrow','broad'))]

ggplot(data=iris) +
    geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species), size=1.3, alpha=.8) +
    stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species), method='lm', size=.2) +
    facet_wrap(~shape) +
    scale_y_continuous(breaks=seq(0,10)) +
    ggtitle('Iris features') +
    theme_bw()
```

```
Warning message in qt((1 - level)/2, df):
"NaNs produced"
```



data

geom

aes

facet

coord

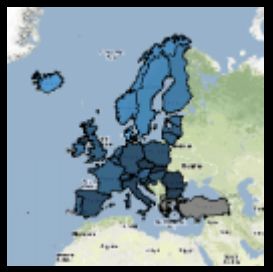
guide

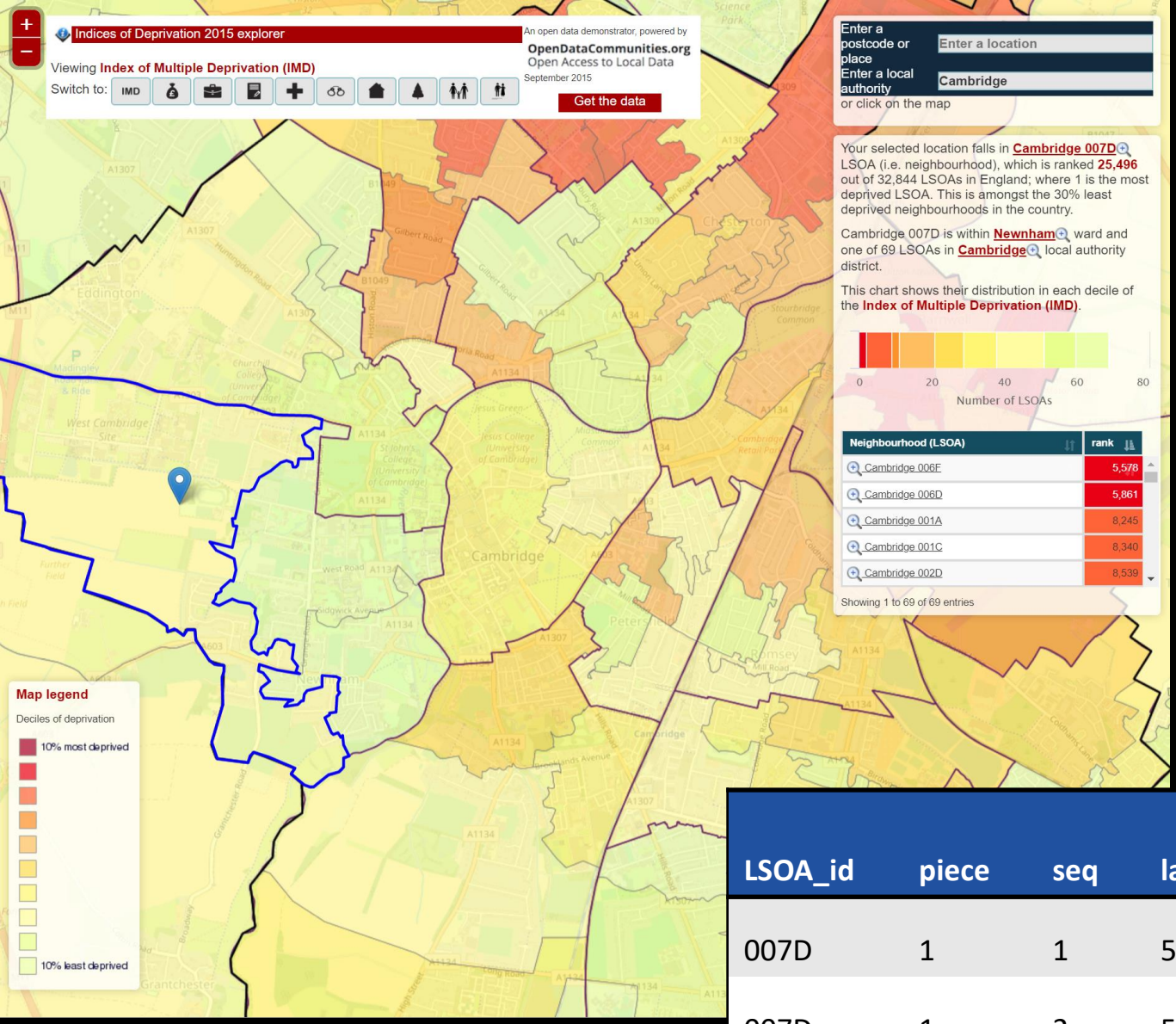
data

Sepal. Length	Sepal. Width	Petal. Length	Petal. Width	Species
5.0	3.4	1.6	0.4	setosa
6.5	3.0	5.5	1.8	virginica
5.0	3.5	1.3	0.3	setosa
6.7	2.5	5.8	1.8	virginica

Data comes in data frames.

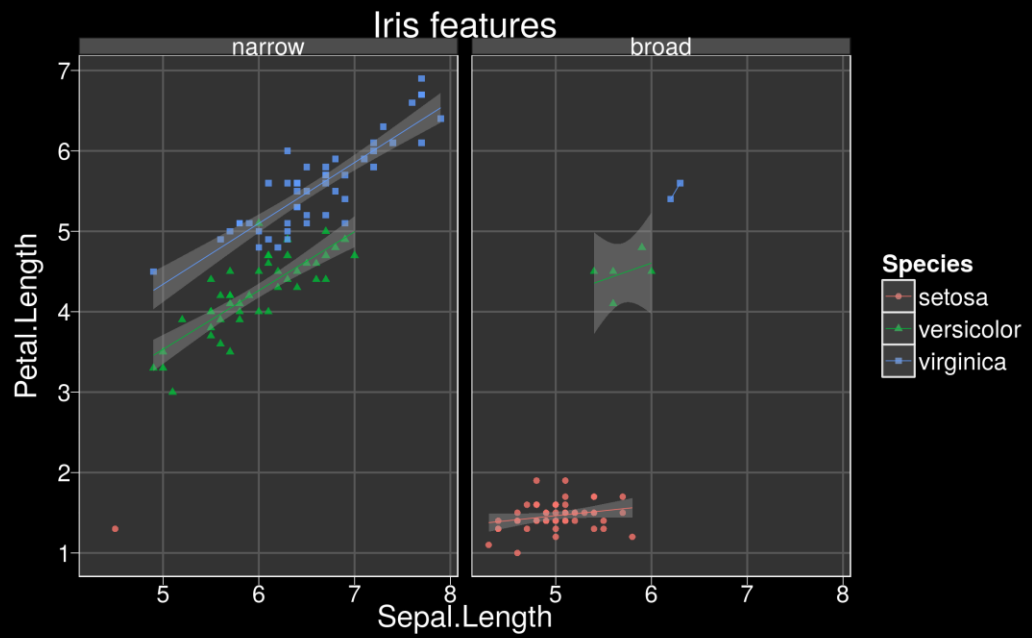
ggplot2 is only for this sort of data (though it has handy functions for *fortifying* some other data types into data frames).





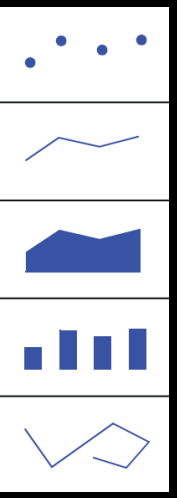
LSOA_id	piece	seq	lat	lng	deprivation
007D	1	1	52.0	1.1	0.4
007D	1	2	52.3	1.05	0.4
007D	1	3	52.1	1.3	0.4
007D	2	4	52.1	1.2	0.4

```
ggplot(data=iris) +
    geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species),
               size=1.3, alpha=.8) +
    stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species),
                method='lm', size=.2) +
    facet_wrap(~shape) +
    scale_y_continuous(breaks=seq(0,10)) +
    ggtitle('Iris features') +
    theme_bw()
```



data. geom. aes. facet. coord. guide.

geom

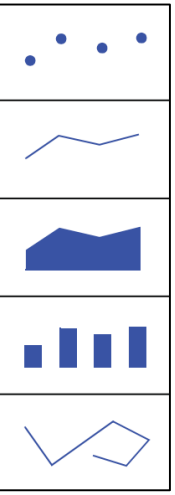


Data is plotted as geoms. A geom occupies part of a coordinate space.

A *geom_bar* occupies an interval of space, and is specified by its two ends (often one is zero). Where the bar is placed is another matter...

gender	response	value	bar	bar
female	rarely	0.08		
female	infrequently	0.11		
female	occasionally	0.17		
female	frequently	0.32		
female	not sure	0.32		
male	rarely	0.30		
male	infrequently	0.15		
male	occasionally	0.10		
male	frequently	0.07		
male	not sure	0.38		

A *geom_bar* occupies an interval of space, and is specified by its two ends (often one is zero). Where the bar is placed is another matter...



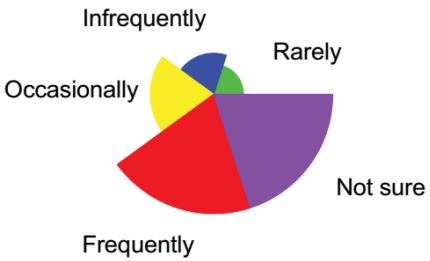
point

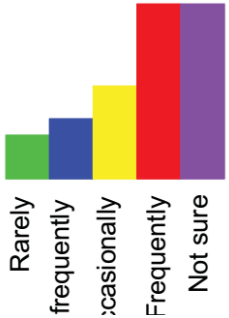
line

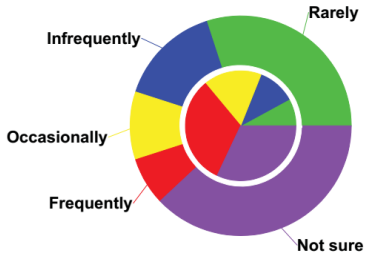
area

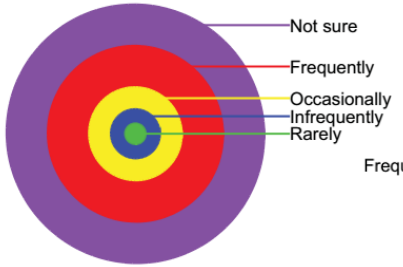
interval

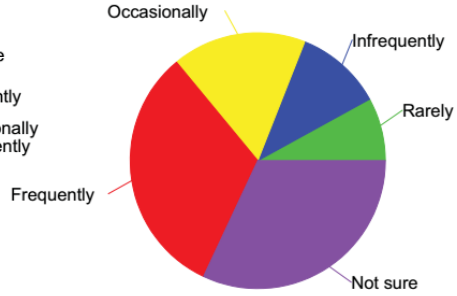
path

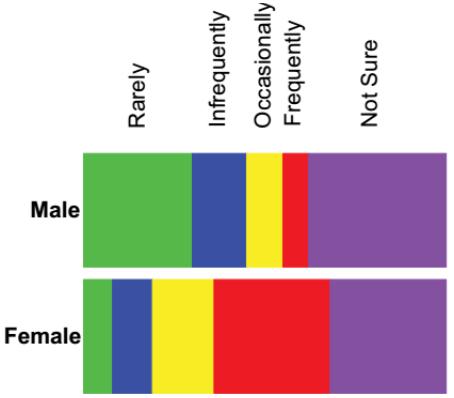




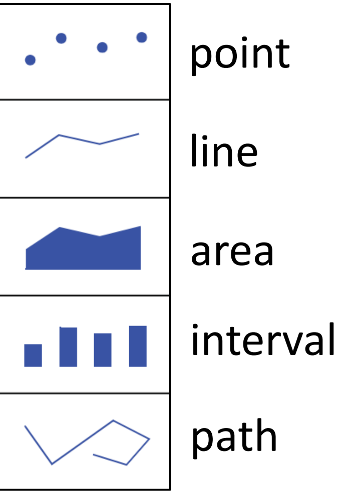


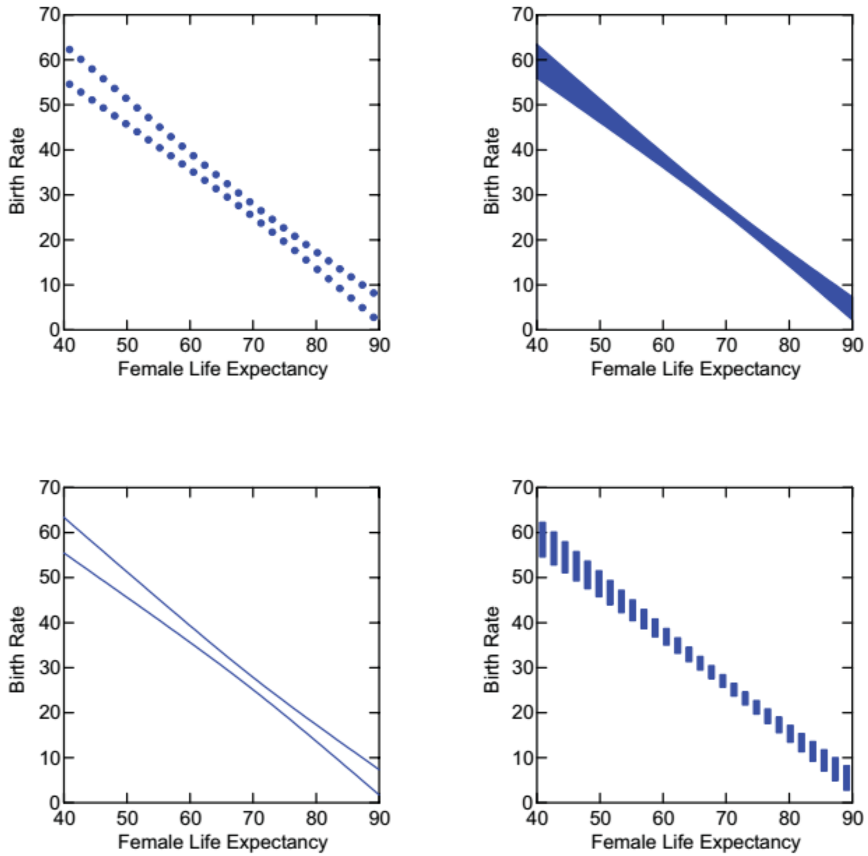






The same data can be represented by many different geoms.

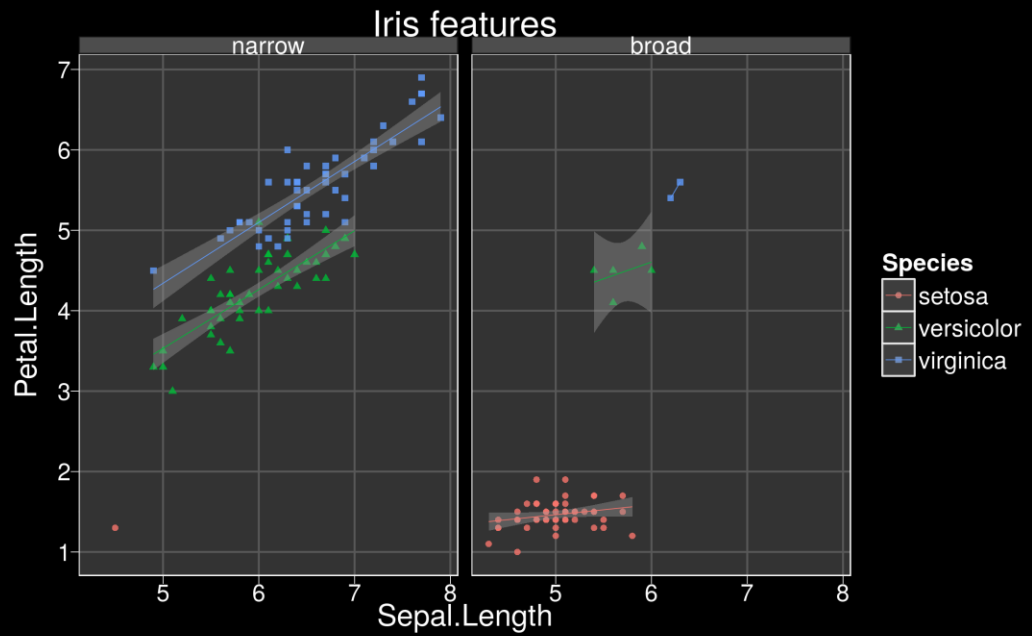




```
ggplot(data=iris) +
    geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species),
               size=1.3, alpha=.8) +
    stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species),
                method='lm', size=.2) +
    facet_wrap(~shape) +
    scale_y_continuous(breaks=seq(0,10)) +
    ggtitle('Iris features') +
    theme_bw()
```

geom_ribbon

The stat_smooth first transforms the data, then it produces a geom — geom_ribbon by default.



Question. What would this plot look like with stat_smooth(geom='bar')?

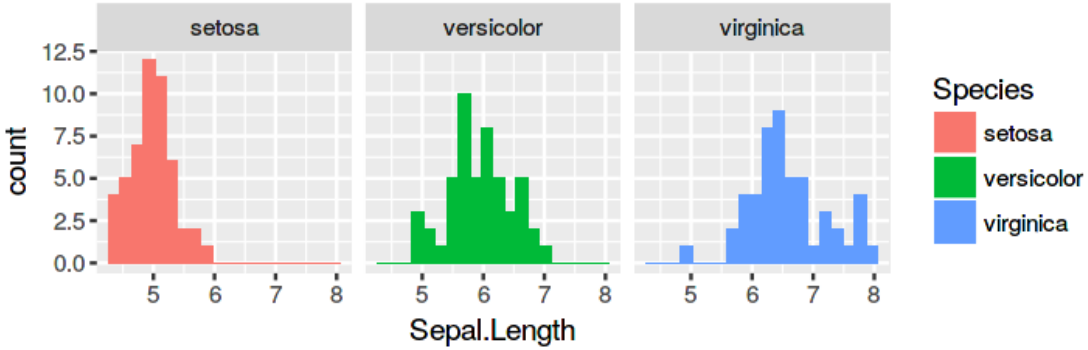
```
ggplot(data=iris) +
    geom_histogram(aes(x=Sepal.Length, fill=Species), bins=20) +
    facet_wrap(~Species)

ggplot(data=iris) +
    geom_bar(aes(x=Sepal.Length, fill=Species), stat='bin', bins=20) +
    facet_wrap(~Species)
```

Geoms can also have a stat, applied by default.

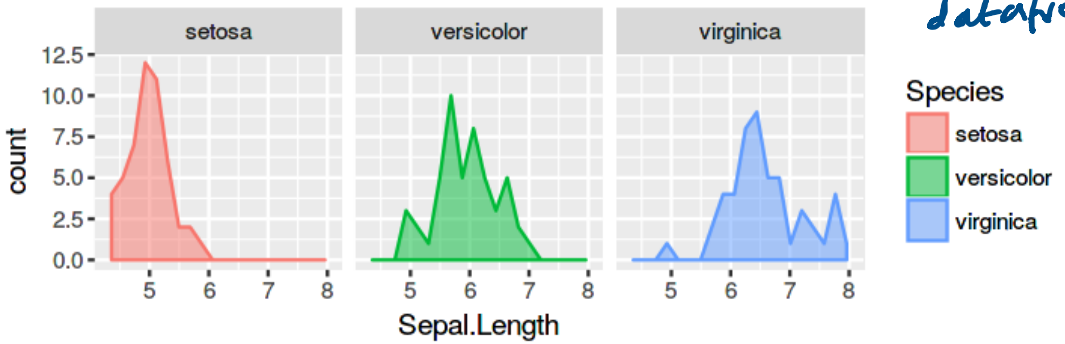
uses stat_bin by default

uses stat_identity by default, but you can always override



a column generated by the stat

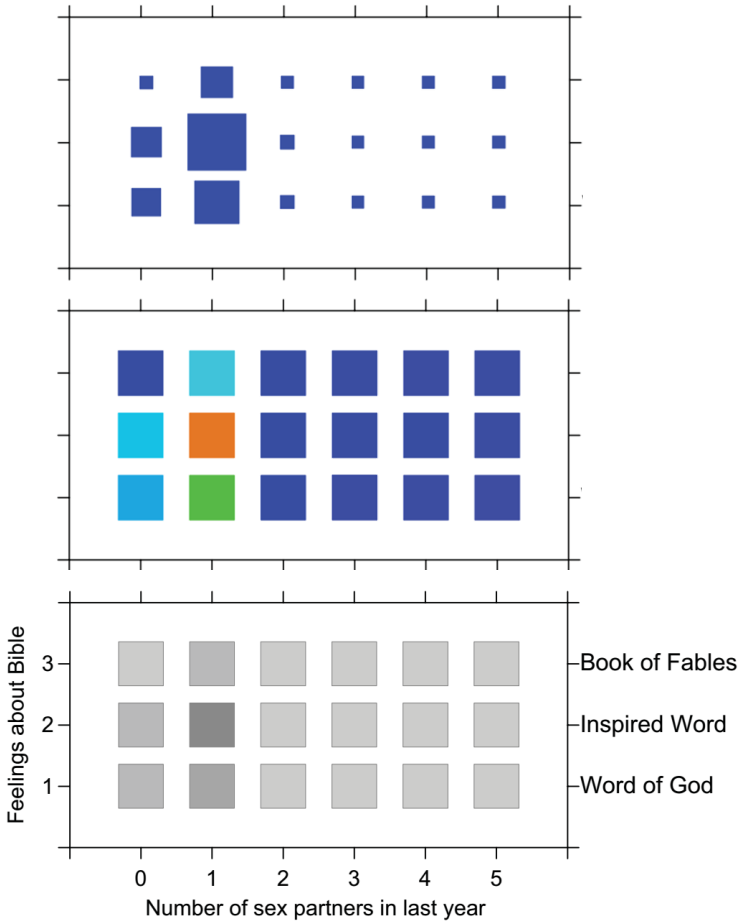
```
ggplot(data=iris) +
    geom_area(aes(x=Sepal.Length, y=..count.., fill=Species, col=Species),
              stat='bin', bins=20, alpha=.5) +
    facet_wrap(~Species)
```



stat_* transforms the dataframe before passing it to geom_*. The new dataframe may have different columns.

An aes is the map from data value to value on some aesthetic scale.

e.g. size, hue, intensity



Data:

feelings about Bible	# sex partners in last year	count
"Word of God"	3	200
⋮	⋮	⋮

Aes:

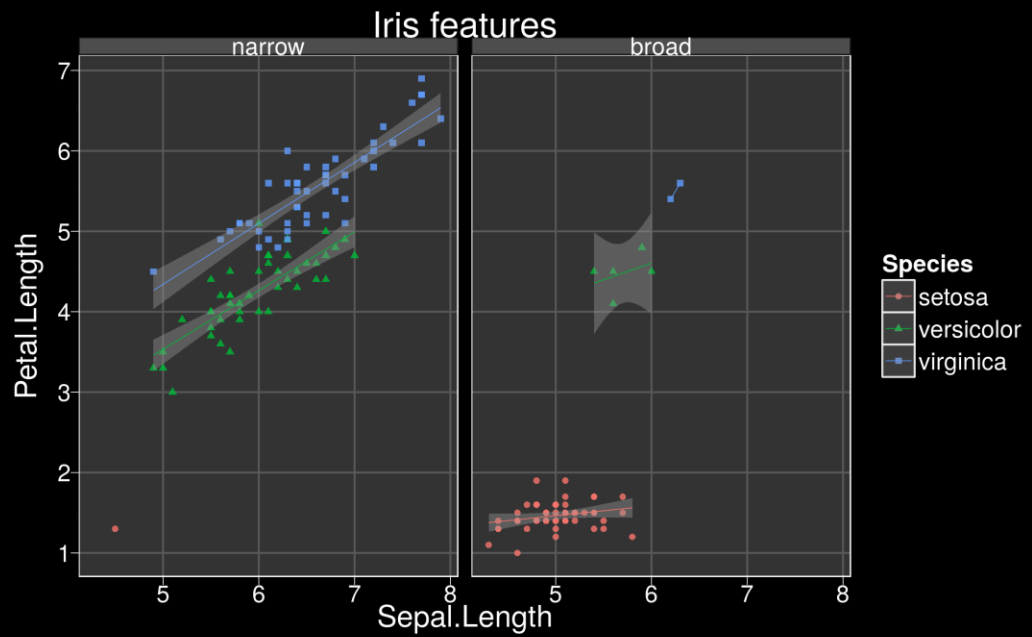
x = '# sex partners in last year'

y = 'feelings about Bible'

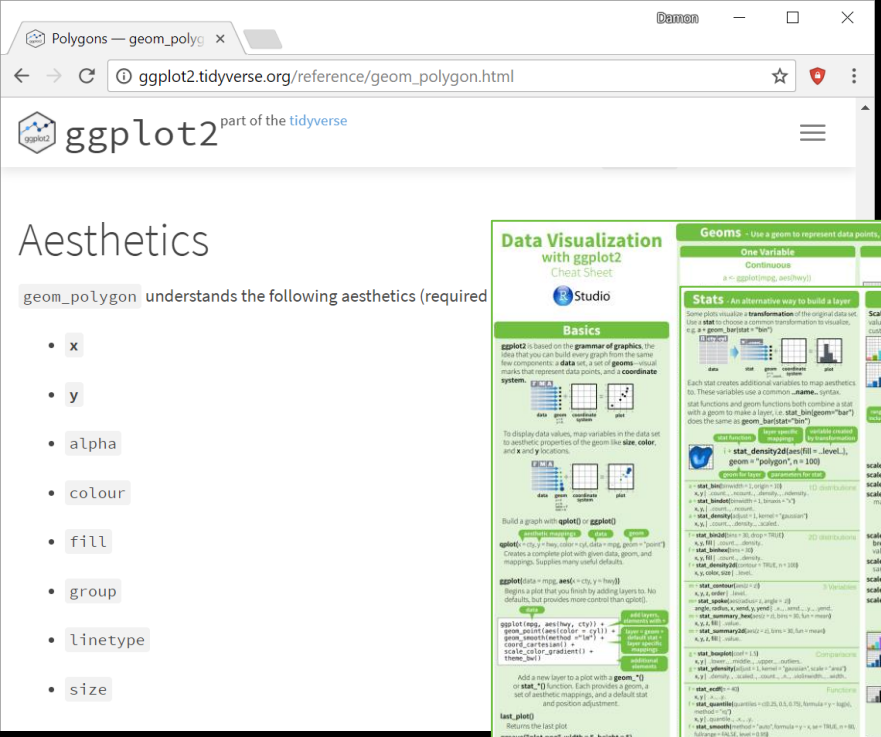
size = count

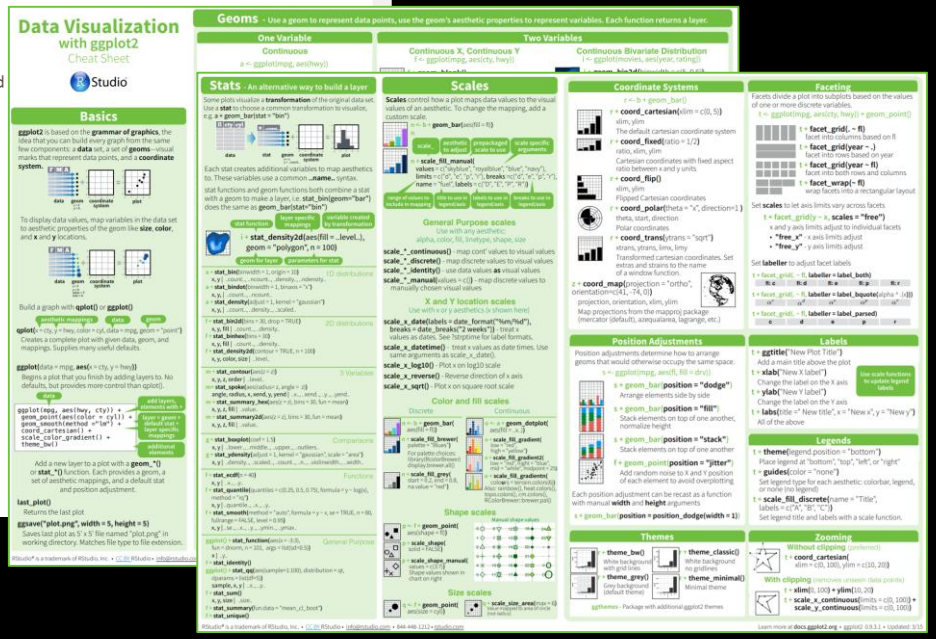
```
ggplot(data=iris) +
    geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species),
               size=1.3, alpha=.8) +
    stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species),
                method='lm', size=.2) +
    facet_wrap(~shape) +
    scale_y_continuous(breaks=seq(0,10)) +
    ggtitle('Iris features') +
    theme_bw()
```

officially this isn't an aes, but morally it is.



data. geom. aes. facet. coord. guide.





Each geom understands a specific set of aes.

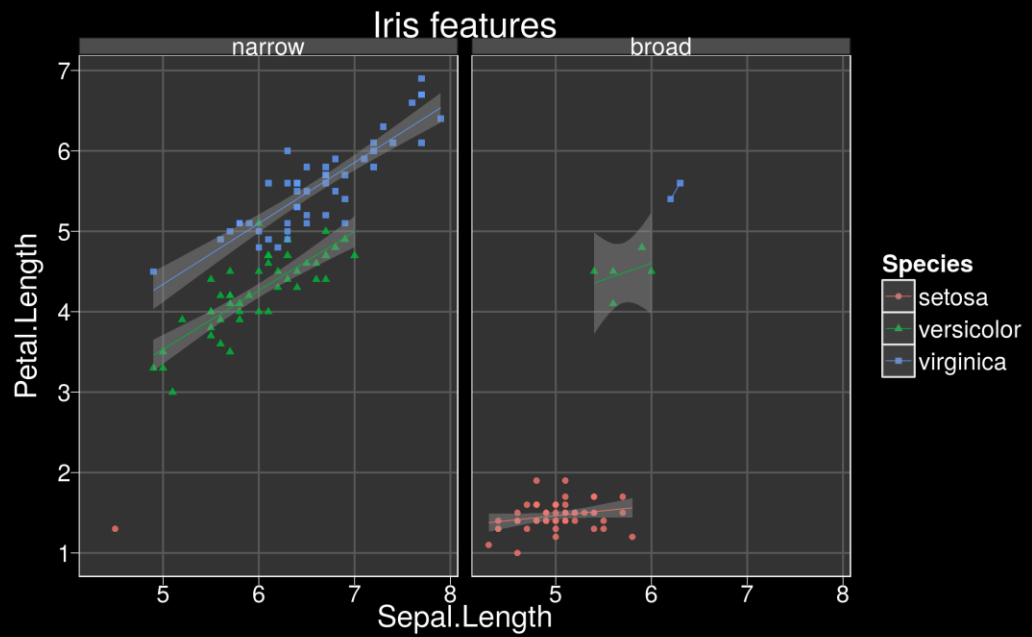
x, y, size. label, shape/pch. fill, colour, alpha. linetype/lty

http://ggplot2.tidyverse.org/reference/

https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

data. geom. **aes.** facet. coord. guide.

```
ggplot(data=iris) +
    geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species),
               size=1.3, alpha=.8) +
    stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species),
                method='lm', size=.2) +
    facet_wrap(~shape) +
    scale_y_continuous(breaks=seq(0,10)) +
    ggtitle('Iris features') +
    theme_bw()
```

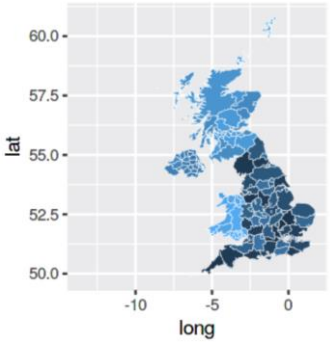


In this plot, these attributes are constants rather than aes.

data. geom. **aes.** facet. coord. guide.

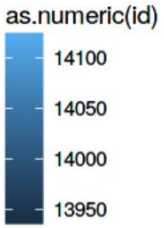
An aes is the map from data value to value on some aesthetic scale.

- The output scale can be customized with scale_*
- It maps the entire range of data values in the entire graphic.

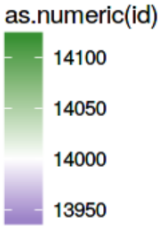



```
ukmap <- fread('https://teachingfiles.blob.core.windows.net/datasets/uk_poly.csv')

ggplot(data=ukmap) +
    geom_polygon(aes(x=long, y=lat, group=group, fill=as.numeric(id)), col='white', size=.1) +
    coord_fixed(ratio=1/cos(50*2*pi/360))
```




```
ggplot(data=ukmap) +
    geom_polygon(aes(x=long, y=lat, group=group, fill=as.numeric(id)), col='white', size=.1) +
    scale_fill_gradient2(midpoint=14000, high='forestgreen', low='darkblue') +
    coord_fixed(ratio=1/cos(50*2*pi/360))
```



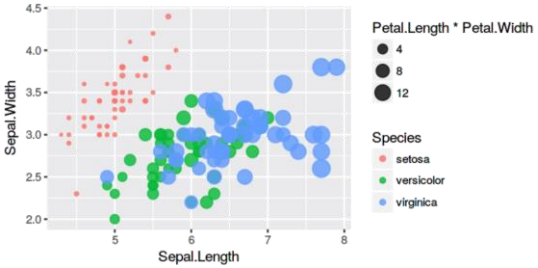
Error: continuous values supplied to a discrete scale

```
ggplot(data=ukmap) +
    geom_polygon(aes(x=long, y=lat, group=group, fill=as.numeric(id)), col='white', size=.1) +
    scale_fill_brewer(type='qual') +
    coord_fixed(ratio=1/cos(50*2*pi/360))
```

An aes is the map from data value to value on some aesthetic scale.

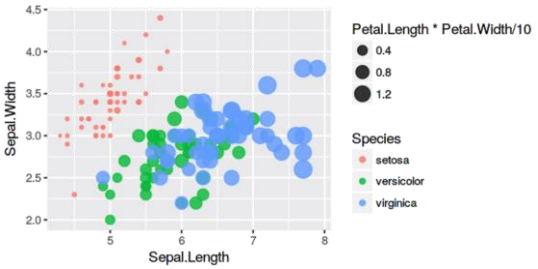
- The output scale can be customized with scale_*
- It maps the entire range of data values in the entire graphic.
  which is a common source of confusion.

```
ggplot(data=iris) +
    geom_point(aes(x=Sepal.Length, y=Sepal.Width, size=Petal.Length * Petal.Width, col=Species), alpha=.8) +
    scale_size_area()
```



Hey, the points are occluding each other.
I need to make them smaller.

```
ggplot(data=iris) +
    geom_point(aes(x=Sepal.Length, y=Sepal.Width, size=Petal.Length * Petal.Width / 10, col=Species), alpha=.8) +
    scale_size_area()
```

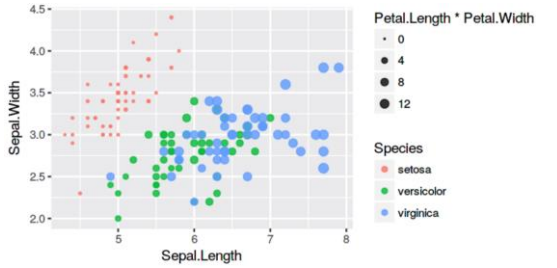


Hold on, the points are exactly the same size as before. WTF?

The data range is different, but scale training has mapped it to precisely the same visual range as before!

```
ggplot(data=iris) +
    geom_point(aes(x=Sepal.Length, y=Sepal.Width, size=Petal.Length * Petal.Width, col=Species), alpha=.8) +
    scale_size_area(max_size=3, limits=c(0,NA))
```

I want to map data range [0, max_in_data] to the output range [0, 3pt]



An aes is the map from data value to value on some aesthetic scale.

- The output scale can be customized with scale_*
- It maps the entire range of data values in the entire graphic.

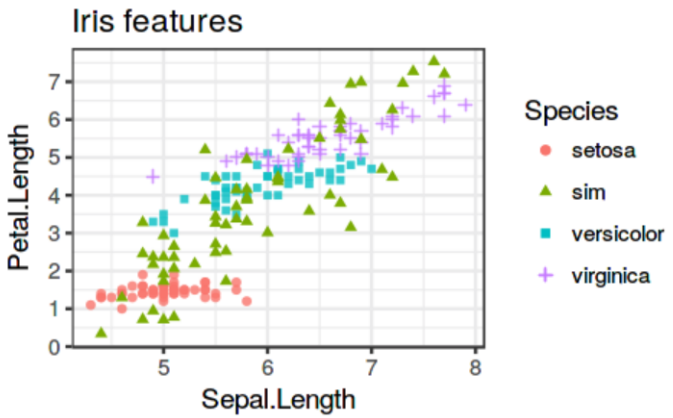
```
fit <- lm(Petal.Length ~ Sepal.Length, data=iris)
df <- copy(iris)
df[, Petal.Length := simulate(fit)]
df <- df[sample(nrow(iris),60,replace=FALSE)]

ggplot(data=iris) +
    geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species), size=1.3, alpha=.8) +
    geom_point(data=df, aes(x=Sepal.Length, y=Petal.Length, col='sim', pch='sim')) +
    scale_y_continuous(breaks=seq(0,10)) +
    ggtitle('Iris features') +
    theme_bw()
```

a new dataset (simulated, from a straight-line fit)

This geom is drawn from iris data

For this geom, we override the dataset, and use df instead.



An aes is the map from data value to value on some aesthetic scale.

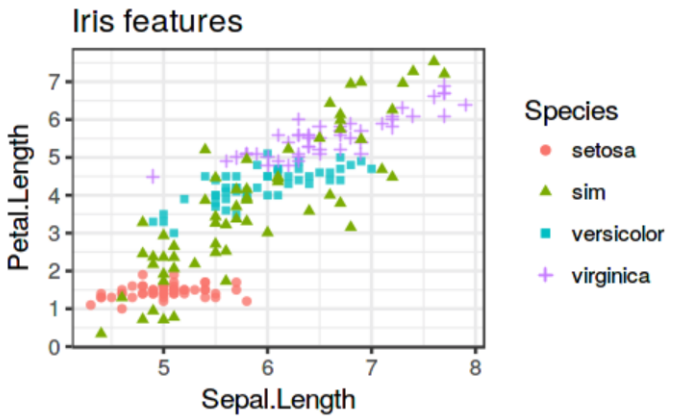
- The output scale can be customized with scale_*
- It maps the entire range of data values in the entire graphic.

```
fit <- lm(Petal.Length ~ Sepal.Length, data=iris)
df <- copy(iris)
df[, Petal.Length := simulate(fit)]
df <- df[sample(nrow(iris),60,replace=FALSE)]

ggplot(data=iris) +
    geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species), size=1.3, alpha=.8) +
    geom_point(data=df, aes(x=Sepal.Length, y=Petal.Length, col='sim', pch='sim')) +
    scale_y_continuous(breaks=seq(0,10)) +
    ggtitle('Iris features') +
    theme_bw()
```

col = 'Species' column from data iris

col = "sim" (a constant) from data df



Training phase:
what data values do we see for col, across the entire plot?
"setosa" "versicolor" "virginica" "sim"

Scale phase:
the data has 4 distinct string values, so we'll use a discrete colour scale by default

This is called the "blending trick"

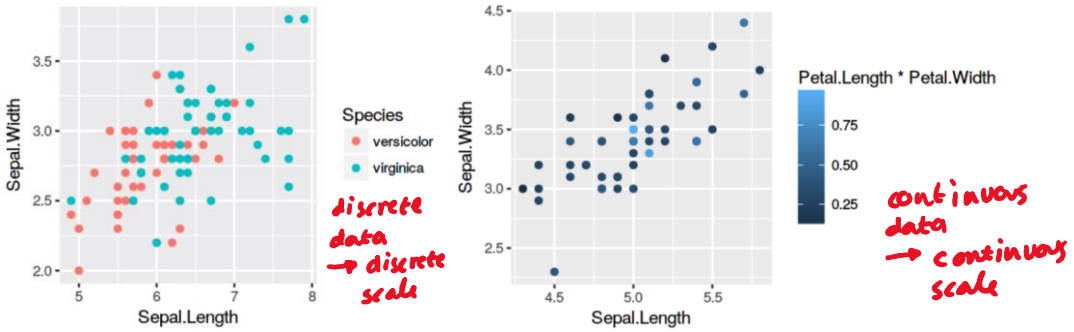
An aes is the map from data value to value on some aesthetic scale.

- The output scale can be customized with scale_*
- It maps the entire range of data values in the entire graphic.

I have two nice plots. What if I combine them?

```
ggplot() +
    geom_point(data=iris[Species!='setosa'], aes(x=Sepal.Length, y=Sepal.Width, col=Species))
```

```
ggplot() +
    geom_point(data=iris[Species=='setosa'], aes(x=Sepal.Length, y=Sepal.Width, col=Petal.Length*Petal.Width))
```



discrete data → discrete scale

continuous data → continuous scale

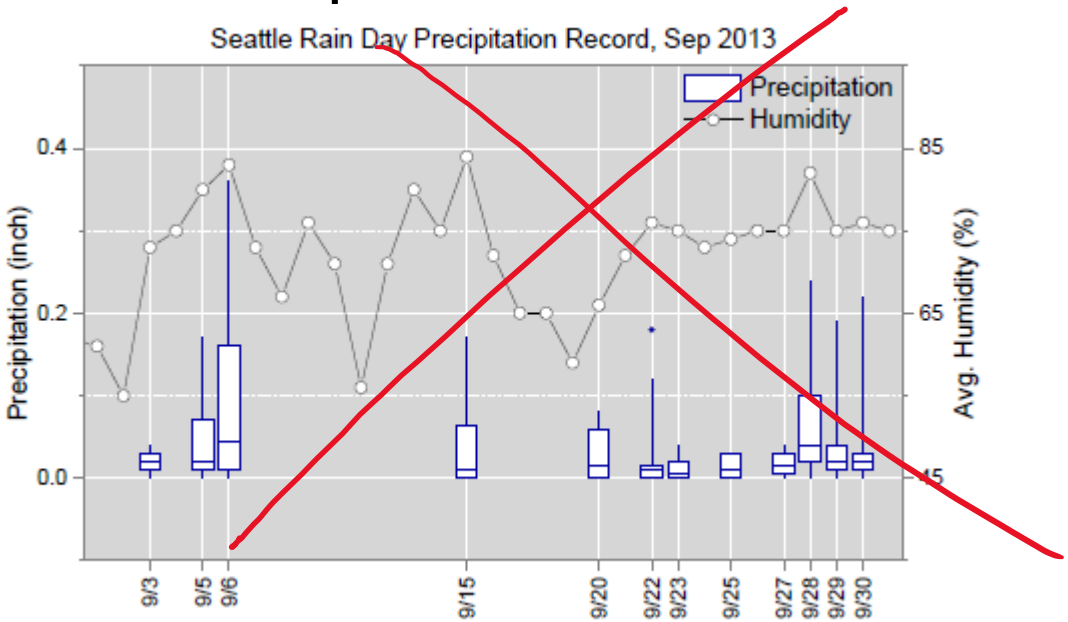
```
ggplot() +
    geom_point(data=iris[Species=='setosa'], aes(x=Sepal.Length, y=Sepal.Width, col=Petal.Length*Petal.Width)) +
    geom_point(data=iris[Species!='setosa'], aes(x=Sepal.Length, y=Sepal.Width, col=Species))
```

This produces an error when it's training the scale.
(In other situations it might coerce num → string and show you a discrete scale with thousands of levels.)

An aes is the map from data value to value on some aesthetic scale.

- The output scale can be customized with scale_*
- It maps the entire range of data values in the entire graphic.

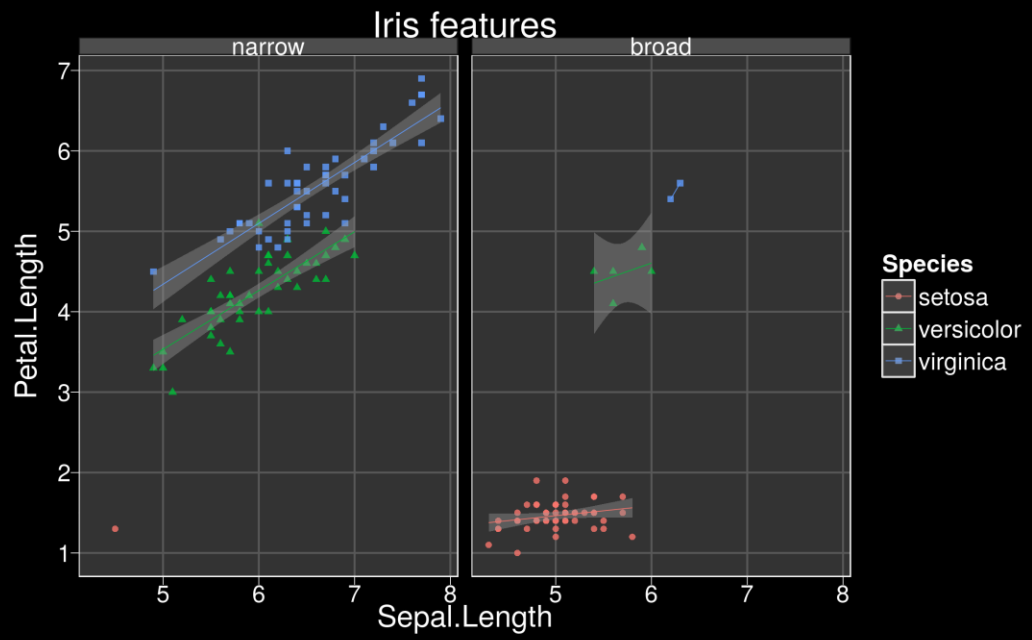
How would you specify the y aes / scale for this plot?



You can't! If you use aes(y = precipitation) & aes(y = Humidity)

ggplot will simply construct a single y scale for all the values.

```
ggplot(data=iris) +
    geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species),
               size=1.3, alpha=.8) +
    stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species),
                method='lm', size=.2) +
    facet_wrap(~shape) +
    scale_y_continuous(breaks=seq(0,10)) +
    ggtitle('Iris features') +
    theme_bw()
```



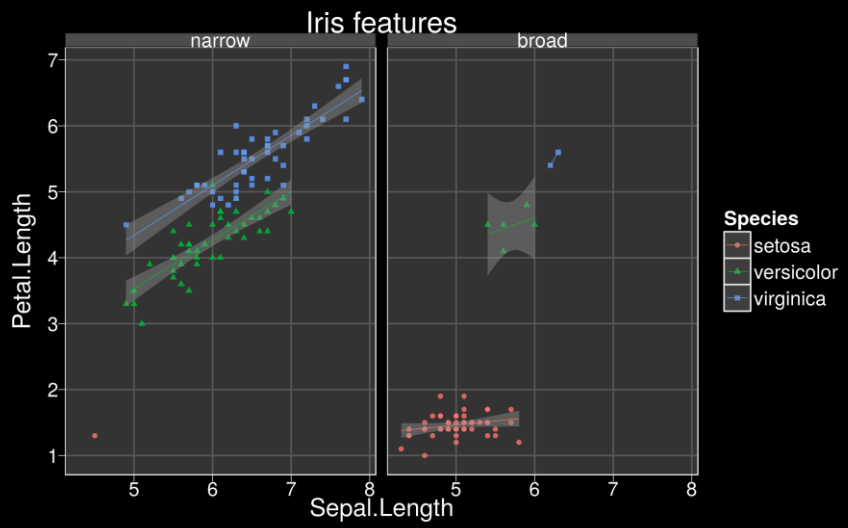
Question. What scale am I customizing in this plot?

Answer. None! The scale_y_continuous() command can be used to customize the scale, e.g. by setting lower and upper ranges (and any data outside those ranges will be mapped to "missing").

But here I'm not controlling the map itself, only how it's displayed.

facet

A facetted plot shows several panels, each containing a subset of the data.

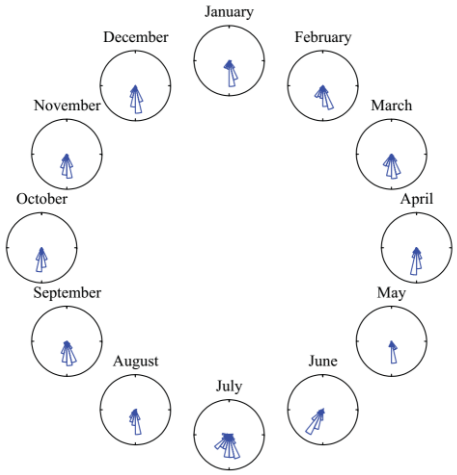



```
ggplot(data=iris) +
    geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species),
               size=1.3, alpha=.8) +
    stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species),
                method='lm', size=.2) +
    facet_wrap(~shape) +
    scale_y_continuous(breaks=seq(0,10)) +
    ggtitle('Iris features') +
    theme_bw()
```

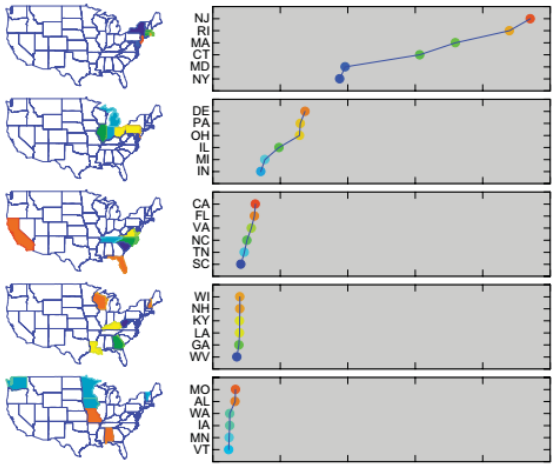
data. geom. aes. **facet.** coord. guide.

A facetted plot shows several panels, each containing a subset of the data.

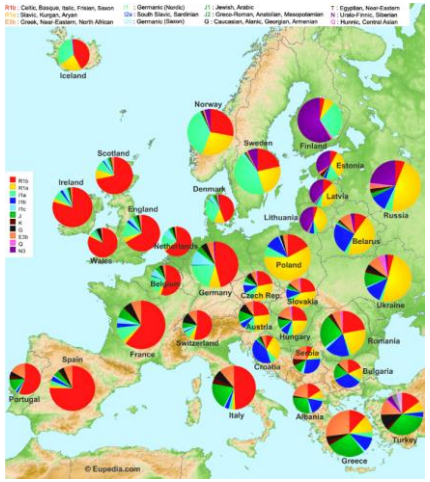
Some things you *can't* do with facets:



The Grammar of Graphics lets you specify how facets are arranged —
but ggplot2 is limited to simple rectangular arrangements.



This is a visual arrangement, not a data arrangement. It's outside the scope of the Grammar of Graphics.
(Hack around with gridExtra instead.)



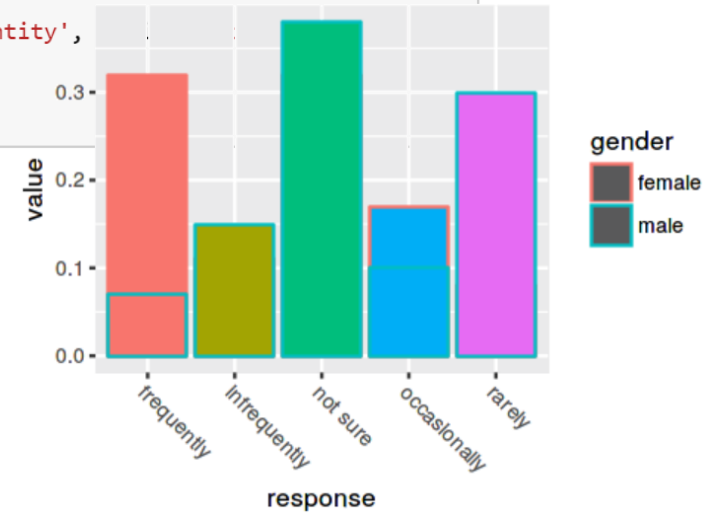
The Grammar of Graphics doesn't go this far (but it should).

coord

The coordinate system says how *x* and *y* aesthetic values should be located on the display.

```
ggplot(data=survey) +
    geom_bar(aes(x=response, y=value, fill=response, col=gender), stat='identity',
    scale_fill_discrete(guide=FALSE) +
    theme_grey(base_size=10) +
    theme(axis.text.x=element_text(angle=-45, hjust=0))
```

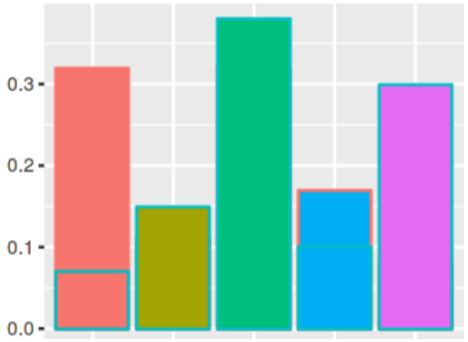
gender	response	value
female	rarely	0.08
female	infrequently	0.11
female	occasionally	0.17
female	frequently	0.32
female	not sure	0.32
male	rarely	0.30
male	infrequently	0.15
male	occasionally	0.10
male	frequently	0.07
male	not sure	0.38



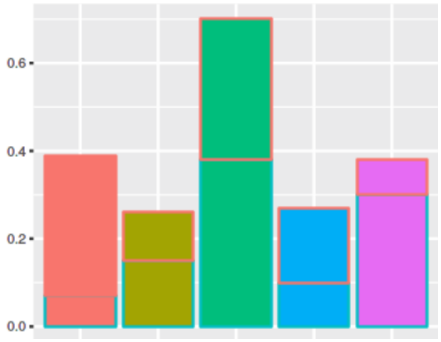
At each x, there are two rows in the table → two overlapping geoms

The coordinate system says how *x* and *y* aesthetic values should be located on the display.

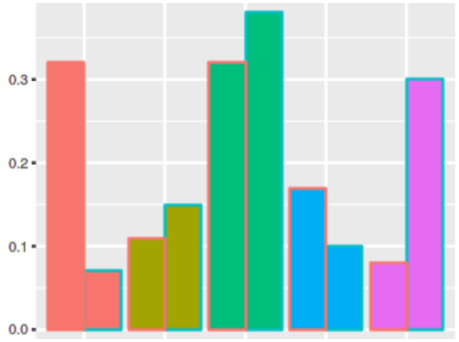
```
ggplot(data=survey) +
    geom_bar(aes(x=response, y=value, fill=response, col=gender), stat='identity', position='stack') +
    scale_fill_discrete(guide=FALSE) +
    theme_grey(base_size=8) +
    theme(axis.text.x=element_text(angle=-45, hjust=0))
```



position_identity



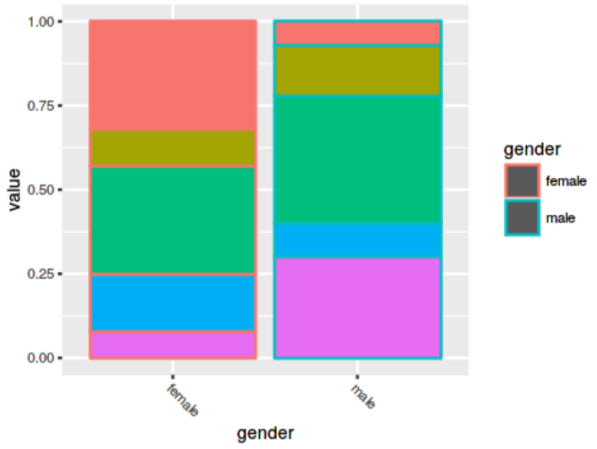
position_stack



position_dodge

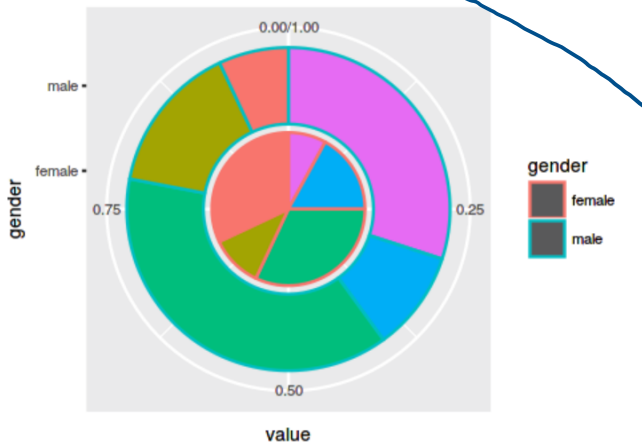
The coordinate system says how *x* and *y* aesthetic values should be located on the display.

```
ggplot(data=survey) +
    geom_bar(aes(x=gender, y=value, fill=response, col=gender), stat='identity', position='stack') +
    scale_fill_discrete(guide=FALSE) +
    theme_grey(base_size=8) +
    theme(axis.text.x=element_text(angle=-45, hjust=0))
```



These are both a stacked bar chart!

```
ggplot(data=survey) +
    geom_bar(aes(x=gender, y=value, fill=response, col=gender), stat='identity', position='stack') +
    scale_fill_discrete(guide=FALSE) +
    coord_polar(theta = "y") +
    theme_grey(base_size=8)
```



use x for radius
use y for angle

The Building Blocks of the Grammar of Graphics:

The Grammar of Graphics is a declarative language for building common data plots. It's great for exploration. It's *not* great for fine art, for animation, for interaction.

data

```
ggplot(data=...)
geom_point(data=...)
stat_smooth()  # transforms the data
geom_bar()     # automatically applies stat_count
```

geom

```
geom_bar()
stat_smooth()  # automatically produces geom_ribbon
```

aes

```
aes(x=...)
scale_fill_gradient2()
```

facet

```
facet_wrap(~...)
facet_grid(...~...)
```

coord

```
position_dodge(), geom_bar(position='dodge')
coord_fixed(ratio=0.4)
coord_cartesian(xlim, ylim)
coord_polar(theta='y')
```

Endless tweaks to the visuals...

guide

```
scale_x_continuous(breaks=...)
scale_fill_gradient2(guide=FALSE)
theme()  gglabel()  ggsave()
```