Intro to grammar of graphics (ggplot2)

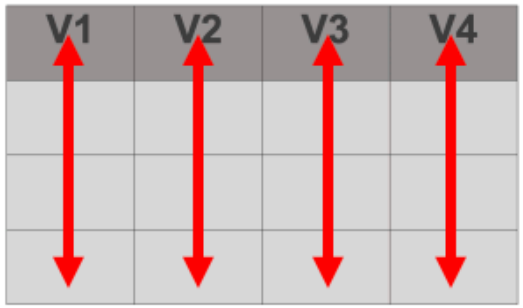
29 January 2020

Modern Research Methods

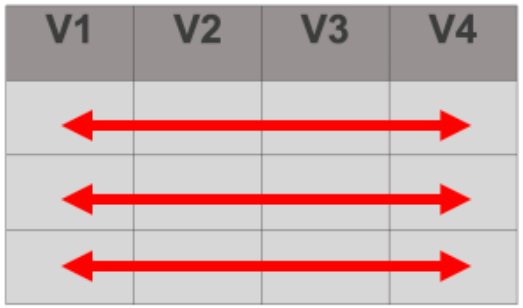
Artwork by @allison_horst

Last Time: Tidy data and deriving information

Each variable is its own column



Each observation is its own row



Each value is its own cell

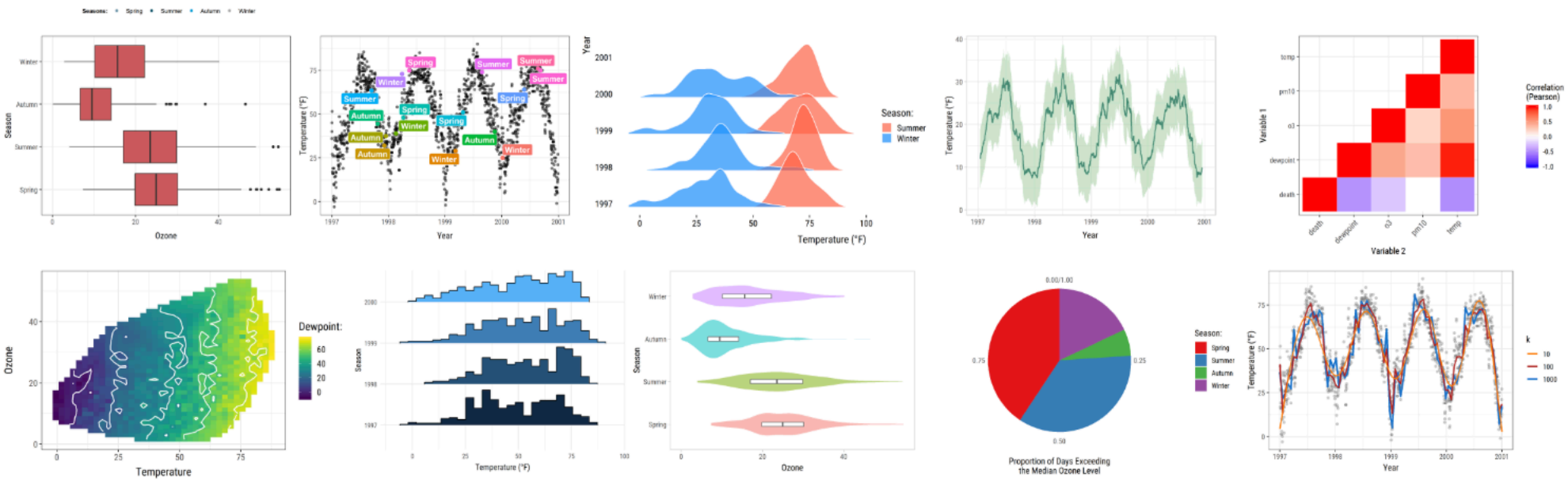
Deriving information

summarise() - summarise **variables**

group_by() - group **cases**

mutate() - create new **variables**

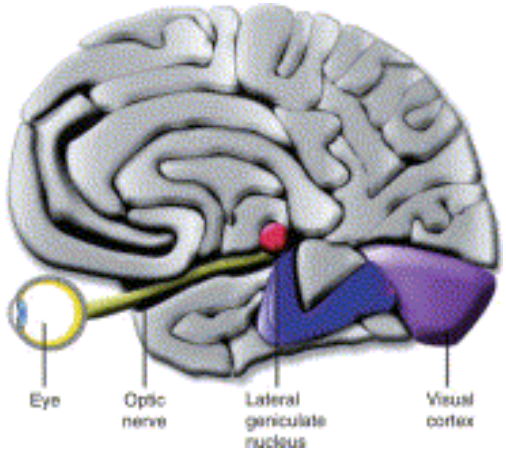
Data Visualization



(image from cedricscherer.netlify.com

Data visualization is the creation and study of the visual representation of data.

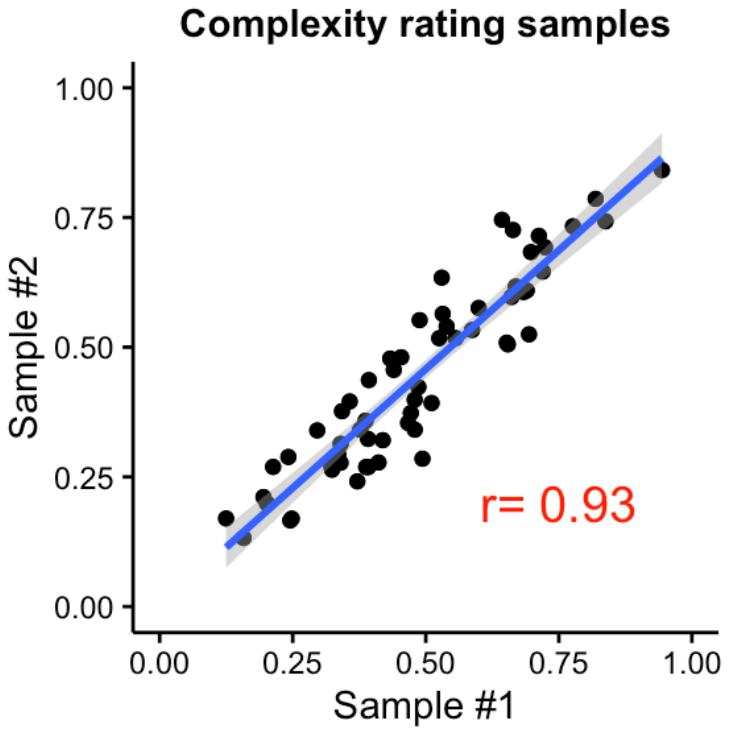
Why do we visualize data? Visual system very quick at identifying patterns and relationships.

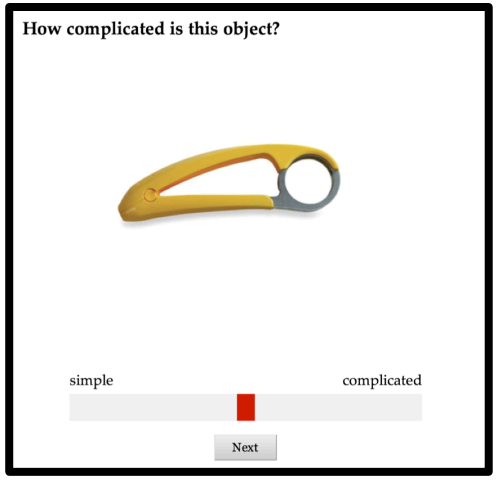


(Rizzi & Bzonamoni, 2012)

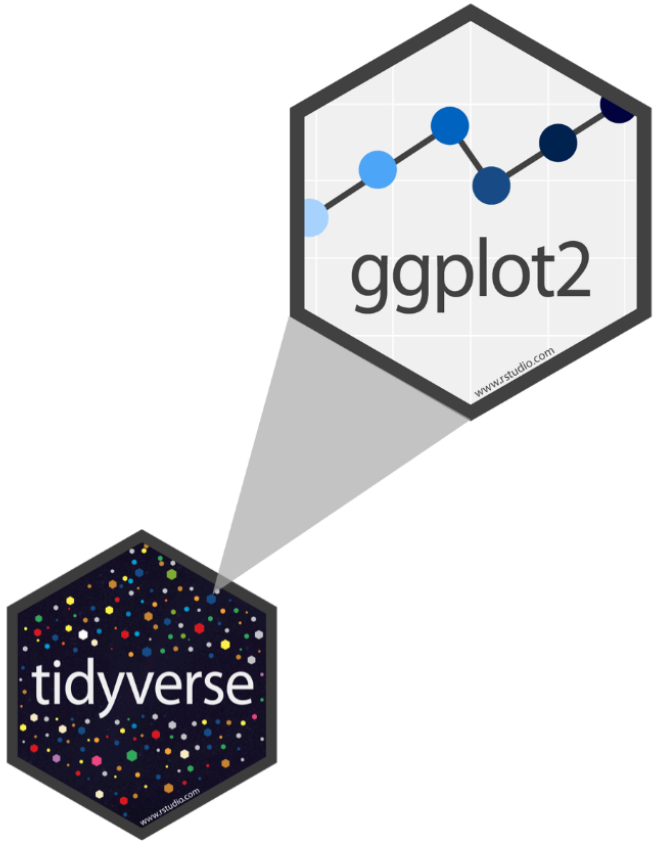
How related are these two variables?

	object_id	sample_1	sample_2
1	1	0.4331551	0.4780250
2	2	0.4112299	0.2776929
3	3	0.1574866	0.1323616
4	4	0.4937611	0.2849934
5	5	0.3914884	0.3230926
6	6	0.3425431	0.3766234
7	7	0.4884821	0.5520592
8	8	0.4191691	0.3206811
9	9	0.3917112	0.2688878
10	10	0.3571429	0.3953984
11	11	0.5317885	0.5642289
12	12	0.4719251	0.3731095
13	13	0.3238265	0.2638602
14	14	0.9438503	0.8412305





ggplot2



- **ggplot2** is tidyverse's data visualization package
- The gg in “ggplot2" stands for Grammar of Graphics
- It is inspired by the book **Grammar of Graphics** by Leland Wilkinson
- A grammar of graphics is a tool that enables us to concisely describe the components of a graphic

What data do you want to plot?

1. Tidy Data

```
p ← ggplot(data = gapminder, ...
```

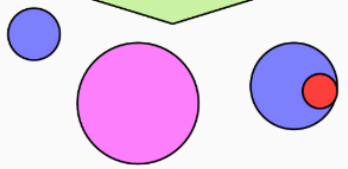
gdp	lifexp	pop	continent
340	65	31	Euro
227	51	200	Amer
909	81	80	Euro
126	40	20	Asia

2. Mapping

Map variables to aesthetics (parts) of plot

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdp,
           y = lifexp, size = pop,
           color = continent))
```

3. Geom



What kind of plot do you want

```
p + geom_point()
```

1. Tidy Data

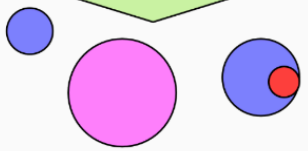
```
p ← ggplot(data = gapminder, ...
```

gdp	lifexp	pop	continent
340	65	31	Euro
227	51	200	Amer
909	81	80	Euro
126	40	20	Asia

2. Mapping

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdp,
           y = lifexp, size = pop,
           color = continent))
```

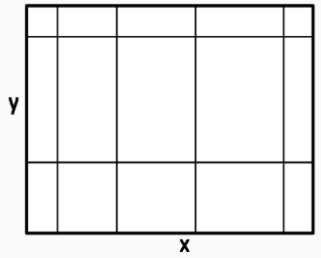
3. Geom



```
p + geom_point()
```

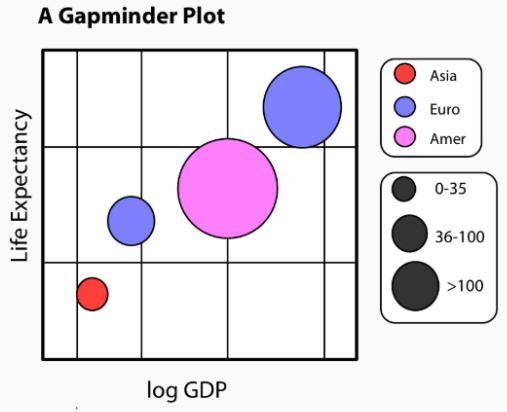
4. Co-Ordinates & Scales

```
p + coord_cartesian() +
    scale_x_log10()
```



5. Labels & Guides

```
p + labs(x = "log GDP",
         y = "Life Expectancy",
         title = "A Gapminder Plot")
```



[Some examples]

General Recipe

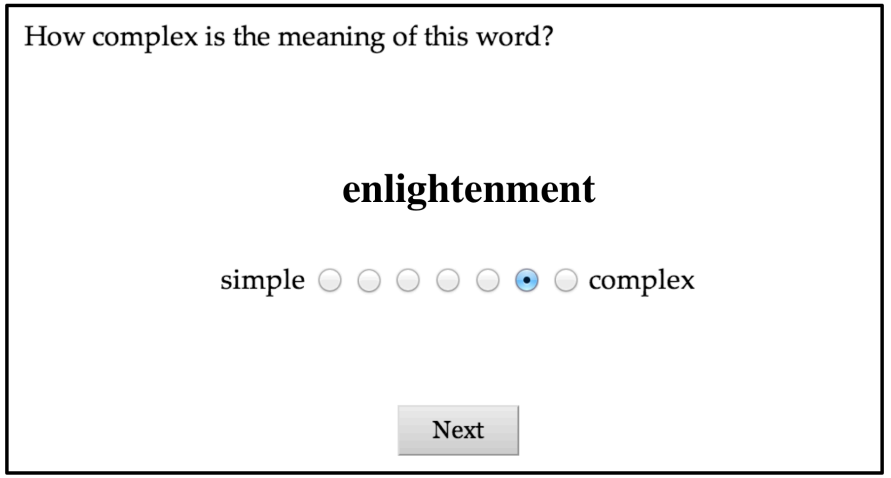
1. Tell the `ggplot()` function what our data is.
2. Tell `ggplot()` *what* relationships we want to see. For convenience we will put the results of the first two steps in an object called `p`.
3. Tell `ggplot` *how* we want to see the relationships in our data.
4. Layer on geoms as needed, by adding them to the `p` object one at a time.
5. Use some additional functions to adjust scales, labels, tick marks, titles. We'll learn more about some of these functions shortly.

```
p <- ggplot(data= <data>,
       mapping= aes(<aesthetic> = <variable>,
                    <aesthetic> = <variable>,
                    <...> = <...>)

p + geom_<type>(<...>)+
    scale_<mapping>_<type>(<...>)+
    coord_<type>(<...>)+
    labs(<...>)
```

Complexity in natural language

Participants rated 499 words for conceptual complexity (Lewis & Frank, 2016)



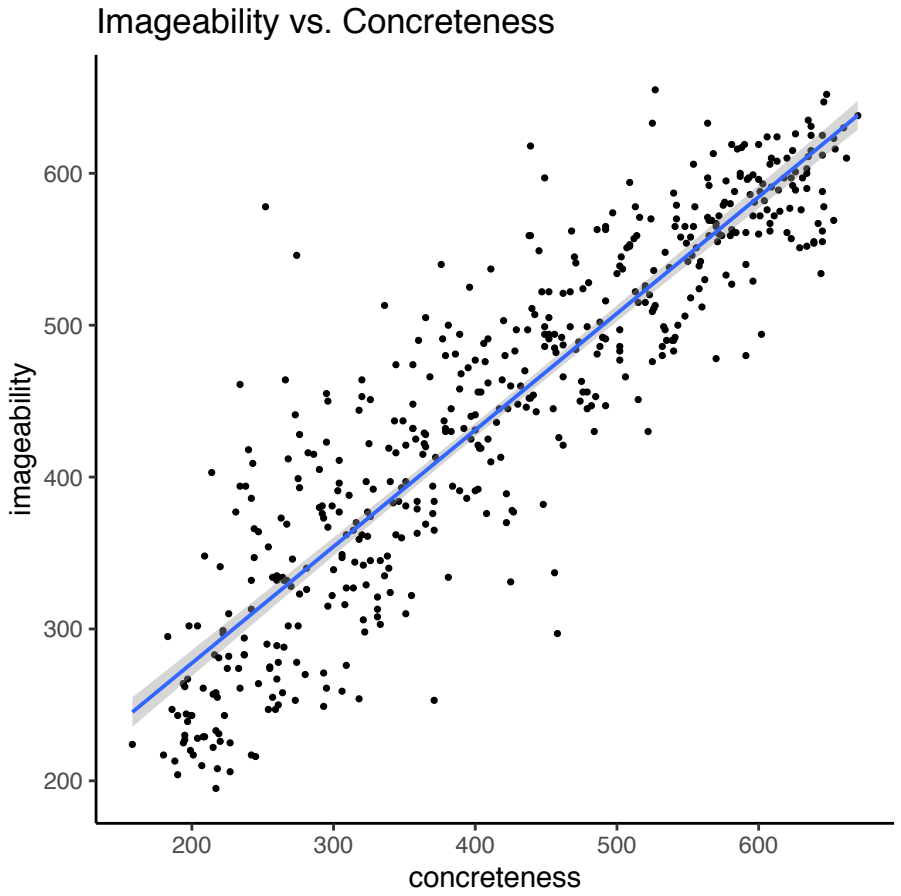
	word	mean_complexity_rating	complexity_type	word_class	n_chars	n_syllables	n_phonemes	concreteness	familiarity	imageability	frequency
1	a	1.375	low	closed	1	1	1	201	632	217	410.55
2	about	3.9375	high	closed	5	2	4	227	593	225	193.97
3	ache	2.88235294117647	low	open	4	1	3	443	523	443	8.23
4	affirmation	5.83333333333333	high	open	11	4	8	242	332	313	4.25
5	after	2.23076923076923	low	closed	5	2	4	242	575	217	93.85
6	age	2.17647058823529	low	open	3	1	2	390	582	468	36.83
7	aid	1.625	low	open	3	1	2	372	536	413	17.31
8	all	2.66666666666667	low	closed	3	1	2	267	582	332	225.97
9	ally	3.23076923076923	low	open	4	2	3	485	410	453	
10	alphabet	3.57142857142857	high	open	8	3	7	449	493	499	8.23

What are some questions we could ask with this data?

	word	mean_complexity_rating	complexity_type	word_class	n_chars	n_syllables	n_phonemes	concreteness	familiarity	imageability	frequency
1	a	1.375	low	closed	1	1	1	201	632	217	410.55
2	about	3.9375	high	closed	5	2	4	227	593	225	193.97
3	ache	2.88235294117647	low	open	4	1	3	443	523	443	8.23
4	affirmation	5.83333333333333	high	open	11	4	8	242	332	313	4.25
5	after	2.23076923076923	low	closed	5	2	4	242	575	217	93.85
6	age	2.17647058823529	low	open	3	1	2	390	582	468	36.83
7	aid	1.625	low	open	3	1	2	372	536	413	17.31
8	all	2.66666666666667	low	closed	3	1	2	267	582	332	225.97
9	ally	3.23076923076923	low	open	4	2	3	485	410	453	
10	alphabet	3.57142857142857	high	open	8	3	7	449	493	499	8.23

What's the relationship between imageability and concreteness?

```
ggplot(word_df,
               aes(x = concreteness,
                   y = imageability)) +
  geom_point(size = 1.2) +
  geom_smooth(method = "lm") +
  ggtitle("Imageability vs. Concreteness") +
  theme_classic(base_size = 16)
```



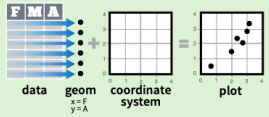
Interpretation: More concrete words tend to be more imageable.

Data Visualization with ggplot2 Cheat Sheet

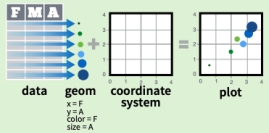
RStudio

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data** set, a set of **geoms**—visual marks that represent data points, and a **coordinate system.**



To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.



Build a graph with **qplot()** or **ggplot()**

aesthetic mappings · data · geom

qplot(x = cty, y = hwy, color = cyl, data = mpg, geom = "point"**)**
Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

ggplot(data = mpg, **aes(**x = cty, y = hwy**))**
Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

data

```
ggplot(mpg, aes(hwy, cty)) +
 geom_point(aes(color = cyl)) +
 geom_smooth(method ="lm") +
 coordinate() +
 scale_color_gradient() +
 theme_bw()
```

add layers, elements with +

layer = geom + default stat + layer specific mappings

additional elements

Add a new layer to a plot with a **geom_*()** or **stat_*()** function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

last_plot()
Returns the last plot

ggsave("plot.png", width = 5, height = 5)
Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

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

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

a + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

a + geom_dotplot()
x, y, alpha, color, fill

a + geom_freqpoly()
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

a + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete

b <- ggplot(mpg, aes(fl))

b + geom_bar()
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

c <- ggplot(map, aes(long, lat))

c + geom_polygon(aes(group = group)**)**
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

d + geom_path(lineend="butt", linejoin="round', linemitre=1**)**
x, y, alpha, color, linetype, size

d + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900**)**
x, ymax, ymin, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

e + geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat**)**
x, xend, y, yend, alpha, color, linetype, size

e + geom_rect(aes(xmin = long, ymin = lat, xmax= long + delta_long, ymax = lat + delta_lat**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

f + geom_blank()

f + geom_jitter()
x, y, alpha, color, fill, shape, size

f + geom_point()
x, y, alpha, color, fill, shape, size

f + geom_quantile()
x, y, alpha, color, linetype, size, weight

f + geom_rug(sides = "bl"**)**
alpha, color, linetype, size

f + geom_smooth(model = lm**)**
x, y, alpha, color, fill, linetype, size, weight

f + geom_text(aes(label = cty)**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + geom_bar(stat = "identity")
x, y, alpha, color, fill, linetype, size, weight

g + geom_boxplot()
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + geom_dotplot(binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill

g + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

h + geom_jitter()
x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

i + geom_bin2d(binwidth = c(5, 0.5)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + geom_density2d()
x, y, alpha, colour, linetype, size

i + geom_hex()
x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

j + geom_area()
x, y, alpha, color, fill, linetype, size

j + geom_line()
x, y, alpha, color, linetype, size

j + geom_step(direction = "hv"**)**
x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

k + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, linetype, size

k + geom_errorbar()
x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)

k + geom_linerange()
x, ymin, ymax, alpha, color, linetype, size

k + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

l + geom_map(aes(map_id = state), map = map**) + expand_limits(**x = map$long, y = map$lat**)**
map_id, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

m + geom_contour(aes(z = z)**)**
x, y, z, alpha, colour, linetype, size, weight

m + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

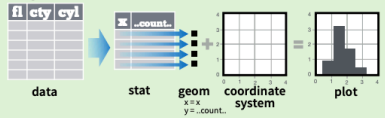
m + geom_tile(aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size

Geoms

Scales

Stats - An alternative way to build a layer

Some plots visualize a **transformation** of the original data set. Use a **stat** to choose a common transformation to visualize, e.g. **a + geom_bar(stat = "bin")**



Each stat creates additional variables to map aesthetics to. These variables use a common **..name..** syntax.

stat functions and geom functions both combine a stat with a geom to make a layer, i.e. **stat_bin(geom="bar")** does the same as **geom_bar(stat="bin")**

stat function | layer specific mappings | variable created by transformation

i + stat_density2d(aes(fill = ..level..), geom = "polygon", n = 100)

geom for layer | parameters for stat

1D distributions

a + stat_bin(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..
a + stat_bindot(binwidth = 1, binaxis = "x")
x, y, | ..count.., ..ncount..
a + stat_density(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

2D distributions

f + stat_bin2d(bins = 30, drop = TRUE)
x, y, fill | ..count.., ..density..
f + stat_binhex(bins = 30)
x, y, fill | ..count.., ..density..
f + stat_density2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

3 Variables

m + stat_contour(aes(z = z))
x, y, z, order | ..level..
m+ stat_spoke(aes(radius= z, angle = z))
angle, radius, x, xend, y, yend | ..x.., ..xend.., ..y.., ..yend..
m + stat_summary_hex(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..
m + stat_summary2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

Comparisons

g + stat_boxplot(coef = 1.5)
x, y | ..lower.., ..middle.., ..upper.., ..outliers..
g + stat_ydensity(adjust = 1, kernel = "gaussian", scale = "area")
x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

Functions

f + stat_ecdf(n = 40)
x, y | ..x.., ..y..
f + stat_quantile(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq")
x, y | ..quantile.., ..x.., ..y..
f + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95)
x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

General Purpose

ggplot() + stat_function(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5))
x | ..y..
f + stat_identity()
ggplot() + stat_qq(aes(sample=1:100), distribution = qt, dparams = list(df=5))
sample, x, y | ..x.., ..y..
f + stat_sum()
x, y, size | ..size..
f + stat_summary(fun.data = "mean_cl_boot")
f + stat_unique()

Scales

Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale.

n <- b + geom_bar(aes(fill = fl))
n

scale_ | aesthetic to adjust | prepackaged scale to use | scale specific arguments

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

General Purpose scales

Use with any aesthetic:
alpha, color, fill, linetype, shape, size

scale_*_continuous() - map cont' values to visual values
scale_*_discrete() - map discrete values to visual values
scale_*_identity() - use data values **as** visual values
scale_*_manual(values = c()) - map discrete values to manually chosen visual values

X and Y location scales

Use with x or y aesthetics (x shown here)

scale_x_date(labels = date_format("%m/%d"), breaks = date_breaks("2 weeks")) - treat x values as dates. See ?strptime for label formats.
scale_x_datetime() - treat x values as date times. Use same arguments as scale_x_date().
scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

Color and fill scales

Discrete

n <- b + geom_bar(aes(fill = fl))
n + scale_fill_brewer(palette = "Blues")
For palette choices: library(RcolorBrewer) display.brewer.all()
n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

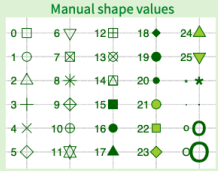
Continuous

o <- a + geom_dotplot(aes(fill = ..x..))
o + scale_fill_gradient(low = "red", high = "yellow")
o + scale_fill_gradient2(low = "red", hight = "blue", mid = "white", midpoint = 25)
o + scale_fill_gradientn(colours = terrain.colors(6))
Also: rainbow(), heat.colors(), topo.colors(), cm.colors(), RColorBrewer::brewer.pal()

Shape scales

p <- f + geom_point(aes(shape = fl))
p + scale_shape(solid = FALSE)
p + scale_shape_manual(values = c(3:7))
Shape values shown in chart on right

Manual shape values



Size scales

q <- f + geom_point(aes(size = cyl))
q + scale_size_area(max = 6)
Value mapped to area of circle (not radius)

Coordinate Systems

r <- b + geom_bar()

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

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

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

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

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

z + coord_map(projection = "ortho", orientation=c(41, -74, 0))
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

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

s + geom_bar(position = "stack")
Stack elements on top of one another

f + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

Each position adjustment can be recast as a function with manual **width** and **height** arguments

s + geom_bar(position = position_dodge(width = 1))

Themes

r + theme_bw()
White background with grid lines

r + theme_grey()
Grey background (default theme)

r + theme_classic()
White background no gridlines

r + theme_minimal
Minimal theme

ggthemes - Package with additional ggplot2 themes

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(. ~ fl)
facet into columns based on fl

t + facet_grid(year ~ .)
facet into rows based on year

t + facet_grid(year ~ fl)
facet into both rows and columns

t + facet_wrap(~ fl)
wrap facets into a rectangular layout

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

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

Set **labeller** to adjust facet labels

t + facet_grid(. ~ fl, labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(. ~ fl, labeller = label_bquote(alpha ^ .(x)))**

α^c	α^d	α^e	α^p	α^r

t + facet_grid(. ~ fl, labeller = label_parsed)

c	d	e	p	r

Labels

t + ggtitle("New Plot Title")
Add a main title above the plot
t + xlab("New X label")
Change the label on the X axis
t + ylab("New Y label")
Change the label on the Y axis
t + labs(title =" New title", x = "New x", y = "New y")
All of the above

Use scale functions to update legend labels

Legends

t + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"
t + guides(color = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)
t + scale_fill_discrete(name = "Title", labels = c("A", "B", "C"))
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred)
t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)
t + xlim(0, 100) + **ylim(**10, 20)
t + scale_x_continuous(limits = c(0, 100)) + **scale_y_continuous(**limits = c(0, 100))

Labels

Themes

How do I save a plot?

```
pdf(“PATH”)
.
.
.
dev.off()
```

```
pdf(“concreteness_plot.pdf”)
ggplot(word_df,
                aes(x = concreteness,
                    y = imageability)) +
  geom_point(size = 1.2) +
  geom_smooth(method = "lm") +
  ggtitle("Imageability vs. Concreteness") +
  theme_classic(base_size = 16)
dev.off()
```

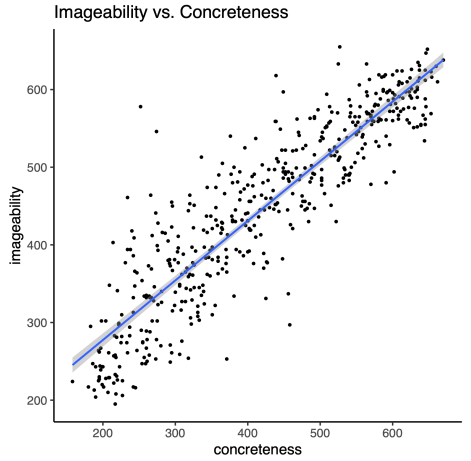
Make your own slide with your own question in groups of 2-3.

- On R studio cloud there is a template with the data in it. ("Class complexity plots - 1-29")
- Use the template to explore one question and make a plot.
- Add the plot to a shared google slide show, linked on the course website under the "Notes" for today's class.

What's the relationship between imageability and concreteness?

```
ggplot(word_df,
              aes(x = concreteness,
                  y = imageability)) +
  geom_point(size = 1.2) +
  geom_smooth(method = "lm") +
  ggtitle("Imageability vs. Concreteness") +
  theme_classic(base_size = 16)
```

Interpretation: More concrete words are more imageable.



Acknowledgements

Slides 5 adopted from https://datasciencebox.org/.

Slides 6-8 adopted from http://socviz.co/makeplot.html#makeplot