# Lecture 10

## Graphics::ggplot II

2018 R Teaching Team

October 3, 2018

# Acknowledgements

1. Mike Fliss & Sara Levintow!
2. stackoverflow (particularly user `David` for lecture styling - link)
3. R Markdown: The Definitive Guide - link Yihui Xie, J. J. Allaire, Garrett Grolemund
4. Hadley for R for Data Scientists and Advanced R
5. R & Rstudio Teams

# This Week

1. **Monday**: Review the `Tidyverse` and understand the basics of `ggplot2`
2. **Today**: Advanced `ggplot2` and `ggplot2` swag

## Structure of Lecture

Contains `R code` in grey boxes and `R output` followed by `##`.

# Namespaces Clarification

You do not need to use `namespaces::` in your code (`ggplot::geom_point`) if you load the package (`library(ggplot)`). I often use namespaces because I forget the exact function name within packages (was it mutate, mutates, etc, etc) and it is preferred practice when writing packages (habits are hard to break). Sorry for the confusion!

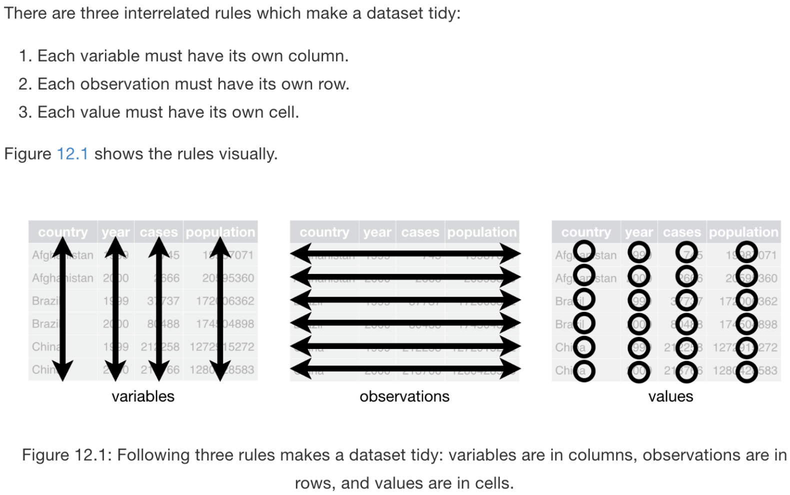
# Note

I have done this to decrease the size of this document. You can do the same and make plots like mine OR do not randomly subset your births dataframe and have plots for your homework.

```r
births <- readRDS(file="<yourpath>")
```

```r
set.seed(43)
births <- births[sample(1:nrow(births), 5e3,
                        replace=FALSE), ]
```

# Let's Review Data Formats (1)



There are three interrelated rules which make a dataset tidy:

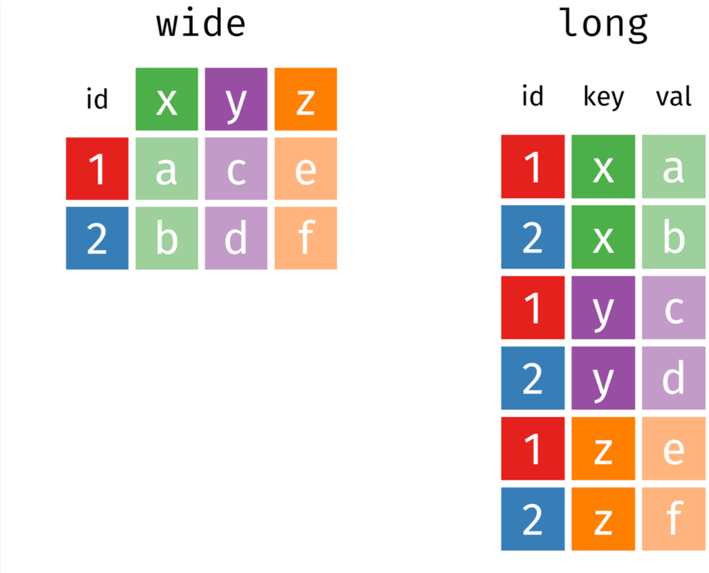
1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

Figure 12.1 shows the rules visually.

Figure 12.1: Following three rules makes a dataset tidy: variables are in columns, observations are in rows, and values are in cells.

Thank you, R for Data Scientists

# Let's Review Data Formats (2)



wide

| id | x | y | z |
|---|---|---|---|
| 1 | a | c | e |
| 2 | b | d | f |

long

| id | key | val |
|---|---|---|
| 1 | x | a |
| 2 | x | b |
| 1 | y | c |
| 2 | y | d |
| 1 | z | e |
| 2 | z | f |

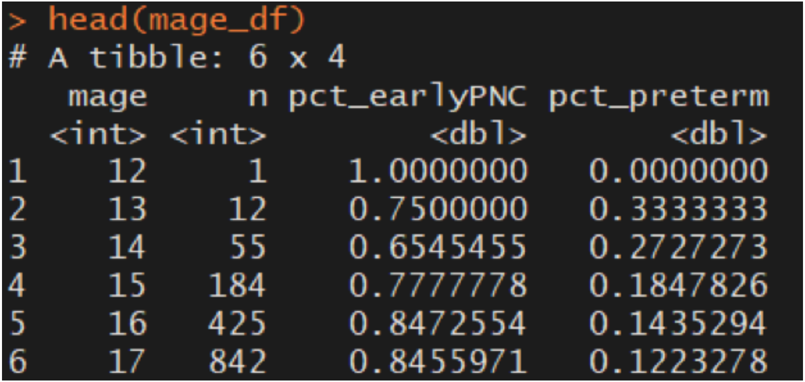
Thank you, Garrick Aden-Buie!

# Let's Review and Practice with the Tidyverse (1)

## Homework 3, Question 2

### Create a summary table for maternal age

**Using dplyr, create a summary table for mage from the births dataset whose head looks like below**



```
> head(mage_df)
# A tibble: 6 x 4
   mage     n pct_earlyPNC pct_preterm
  <int> <int>        <dbl>       <dbl>
1    12     1    1.0000000   0.0000000
2    13    12    0.7500000   0.3333333
3    14    55    0.6545455   0.2727273
4    15   184    0.7777778   0.1847826
5    16   425    0.8472554   0.1435294
6    17   842    0.8455971   0.1223278
```

# Let's Review and Practice with the Tidyverse (2)

Homework 3, Question 2

Pseudocode
**on board**

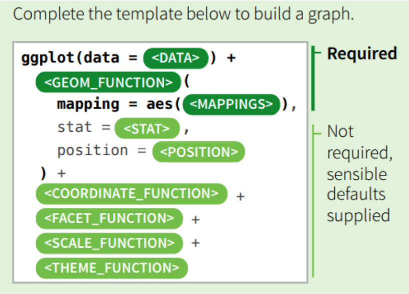
Real Code
**on board**

# Run It

We are going to use this newly created `data.frame` in a few slides.

# Let's Review ggplot (1)

- Grammar of Graphics.
- 5 Main Elements (right).
- Works in layers (appendable).



Complete the template below to build a graph.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

**Required**

Not required, sensible defaults supplied

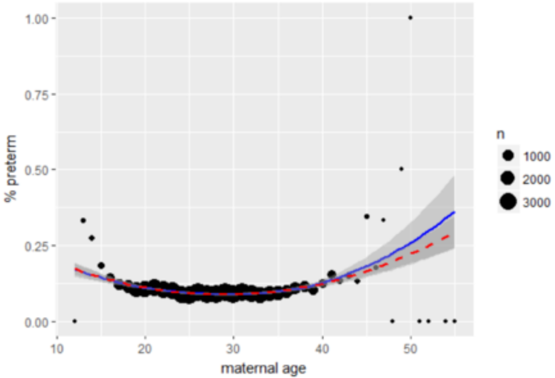
# Let's Review ggplot (2)

## Homework 3, Question 3A

**Create the maternal age functional form plot below (note the "weight" aesthetic is important for LOESS).** *HINT: We are using the mage_df we made above*

% Preterm vs. Maternal Age

Investigating function form of maternal age



Note for future glms: Seems quadratic. Blue is loess, red is square linear.

# Let's Review ggplot (3)

### Homework 3, Question 3A

### Pseudocode

**on board**

### Real Code

**on board**

# Let's Review ggplot (4)

```r
ggplot(data = mage_df, aes(x = mage, y = pct_preterm))+
  geom_point(aes(size=n))+
  geom_smooth(aes(weight=n), color="blue", method="loess")+
  labs(title="% Preterm vs. Maternal Age",
       x="maternal age", y="% preterm",
       subtitle="Investigating function form of maternal age",
       caption="Note for future glms: Seems quadratic. Blue is loess, red is square linear.")
```

# Let's Review ggplot (5)

### Aeshetics

How do I know which `geoms` understand specific `aesthetics`?
`?<geom>`

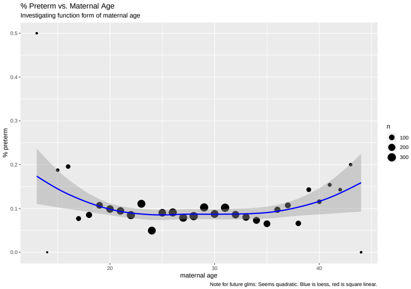
### Stat Transformations

What `method="loess"` is doing behind the scenes is (essentially) running `stat::loess`, creating a new dataframe of "predictions", and overlaying it on our plot. This is to say, we are performing a statistical transformation of our data (i.e. loess regression) and appending the result on as another layer.

# Let's Review ggplot (6)

## Equivalent Approach

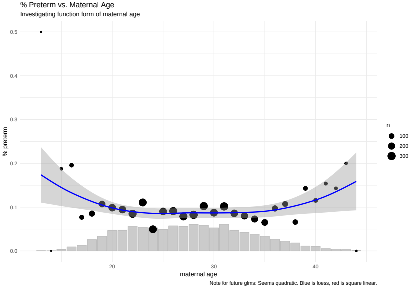
```r
ggplot()+
  geom_point(data=mage_df,
             aes(x=mage, y=pct_preterm, size=n))+
  geom_smooth(data=mage_df,
             aes(x=mage, y=pct_preterm, weight=n),
             color="blue", method="loess")
# + labels (removed for space)
```



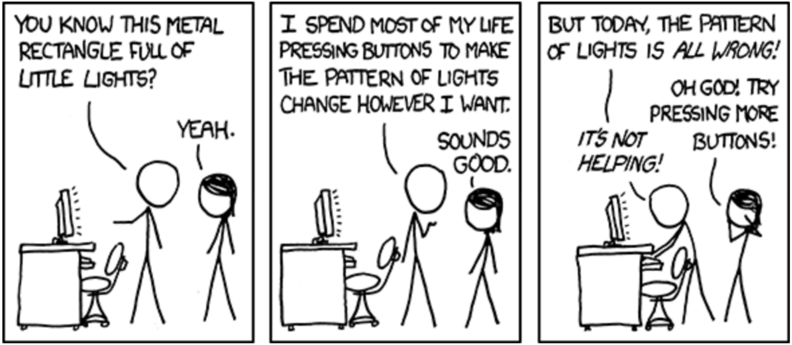
# Let's Review ggplot (7)

## Why use data specific calls for geoms

```
ggplot() +
  geom_bar(data=births, aes(x=mage, y=..prop..),
           color="grey", fill="grey", alpha=0.8) +
  geom_point(data=mage_df,
             aes(x=mage, y=pct_preterm, size=n))+
  geom_smooth(data=mage_df,
             aes(x=mage, y=pct_preterm, weight=n),
             color="blue", method="loess")
# + labels (removed for space)
# # theme change for viewing
```



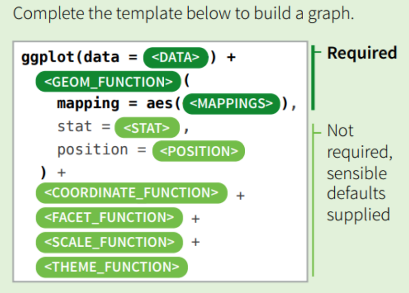
# Check-in


YOU KNOW THIS METAL RECTANGLE FULL OF LITTLE LIGHTS?
YEAH.
I SPEND MOST OF MY LIFE PRESSING BUTTONS TO MAKE THE PATTERN OF LIGHTS CHANGE HOWEVER I WANT.
SOUNDS GOOD.
BUT TODAY, THE PATTERN OF LIGHTS IS ALL WRONG!
OH GOD! TRY PRESSING MORE BUTTONS!
IT'S NOT HELPING!

# Additonal Important Features of ggplot

## Today

- *Coordinate systems* if don't want to use default cartesian coordinates.
- *Facets* to divide a plot into subplots based on the values of one or more discrete variables.
- *Scales* to adjust aesthetics and colors (`scale_*_*()`)
- *Themes* to modify the overall appearance of the plot (background, grid lines).



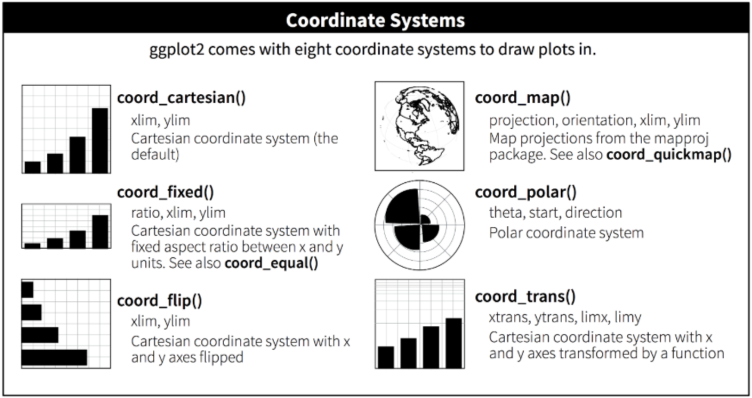
Complete the template below to build a graph.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

**Required**

Not required, sensible defaults supplied

# Coordinate Systems(1)



## Coordinate Systems

ggplot2 comes with eight coordinate systems to draw plots in.

**coord_cartesian()**
xlim, ylim
Cartesian coordinate system (the default)

**coord_fixed()**
ratio, xlim, ylim
Cartesian coordinate system with fixed aspect ratio between x and y units. See also **coord_equal()**

**coord_flip()**
xlim, ylim
Cartesian coordinate system with x and y axes flipped

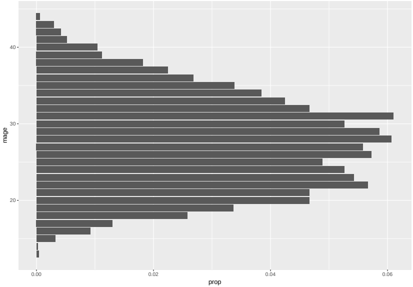
**coord_map()**
projection, orientation, xlim, ylim
Map projections from the mapproj package. See also **coord_quickmap()**

**coord_polar()**
theta, start, direction
Polar coordinate system

**coord_trans()**
xtrans, ytrans, limx, limy
Cartesian coordinate system with x and y axes transformed by a function

# Coordinate Systems (2)

```r
ggplot() +
  geom_bar(data=births, aes(x=mage, y=..prop..)) +
  coord_flip()
```

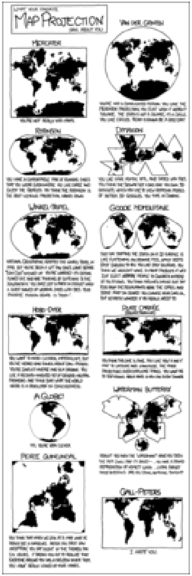


# Map Projections (1)

- Mike will be covering maps the week of 10/29.
- There are a lot of map packages in R (`spplot`, `sp`, `leaflet`, `rgdal`, `tmap`, `ggmap`)
- A new package called `sf` is based on the tidy-data framework (and seems to be the preferred method as of late)
- ggplot: `geom_polygon` (`many`) or `geom_sf` (`sf`)

# Map Projections (2)

Obligatory XKCD Joke.



# Facets

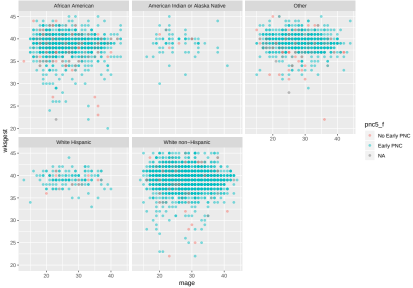
Divide a plot into subplots based on the values of one or more discrete variables. There are two main facet functions:

- facet_wrap
- facet_grid

# facet_wrap

`?facet_wrap`: "facet_wrap wraps a 1d sequence of panels into 2d. This is generally a better use of screen space than facet_grid() because most displays are roughly rectangular." In simpler terms, `facet_wrap` will try to maximize white space for you.

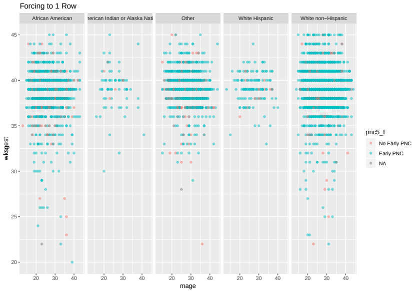
```r
ggplot() +
  geom_point(data=births,
             aes(x=mage, y=wksgest,
                 color=pnc5_f), alpha=0.5) +
  facet_wrap(~raceeth_f)
```



# facet_grid (1)

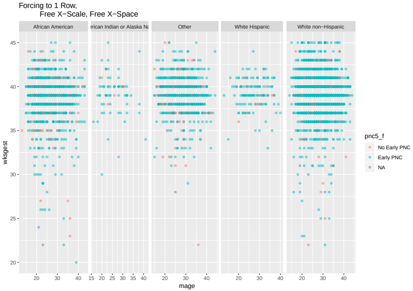
`facet_grid` forms a matrix of plots per your spefications.

```r
ggplot() +
  geom_point(data=births,
             aes(x=mage, y=wksgest,
                 color=pnc5_f), alpha=0.5) +
  facet_grid(.~raceeth_f) +
  ggtitle("Forcing to 1 Row")
```



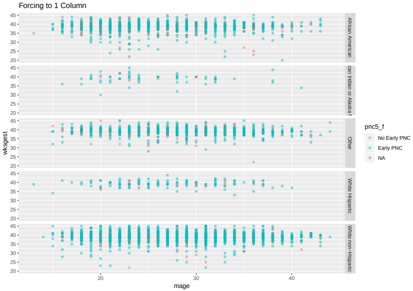
# facet_grid (2)

```
ggplot() +
  geom_point(data=births,
             aes(x=mage, y=wksgest,
                 color=pnc5_f), alpha=0.5) +
  facet_grid(.~raceeth_f,
             scales = "free_x", space="free_x") +
  ggtitle("Forcing to 1 Row,
         Free X-Scale, Free X-Space")
```



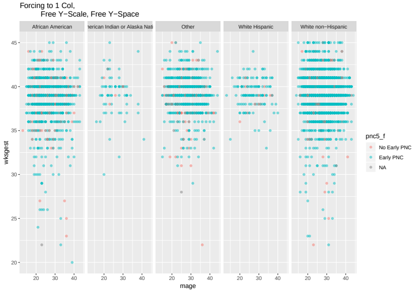
# facet_grid (3)

```
ggplot() +
  geom_point(data=births,
             aes(x=mage, y=wksgest,
                 color=pnc5_f), alpha=0.5) +
  facet_grid(raceeth_f~.) +
  ggtitle("Forcing to 1 Column")
```



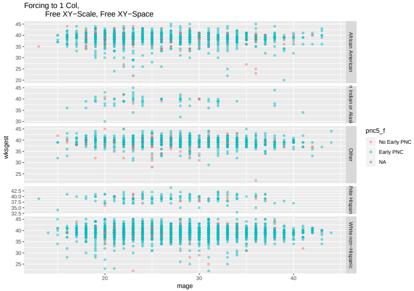
# facet_grid (4)

```
ggplot() +
  geom_point(data=births,
             aes(x=mage, y=wksgest,
                 color=pnc5_f), alpha=0.5) +
  facet_grid(raceeth_f~.,
             scales = "free_y", space="free_y") +
  ggtitle("Forcing to 1 Col,
          Free Y-Scale, Free Y-Space")
```



# facet_grid (5)

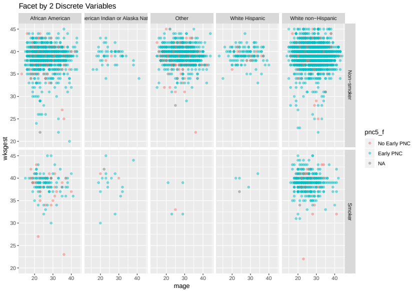
```
ggplot() +
  geom_point(data=births,
             aes(x=mage, y=wksgest,
                 color=pnc5_f), alpha=0.5) +
  facet_grid(raceeth_f~.,
             scales = "free", space="free") +
  ggtitle("Forcing to 1 Col,
         Free XY-Scale, Free XY-Space")
```



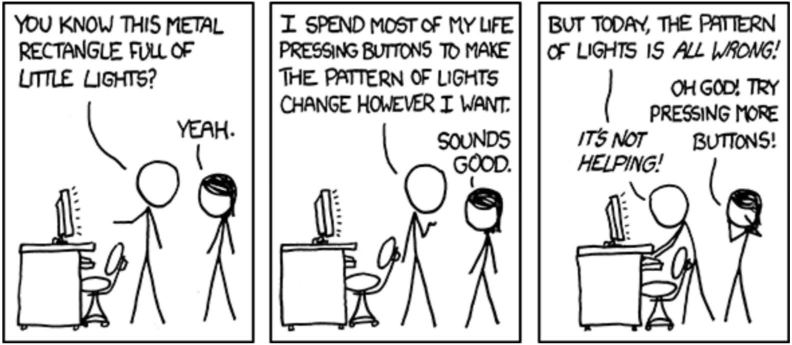
# facet_grid (6)

`facet_grid` by 2 discrete variables.

```
births %>%
  mutate(cigdur_f = factor(cigdur,
                           levels=c(0,1),
                           labels=c("Non-smoker",
                                    "Smoker"))) %>%
  filter(!is.na(cigdur_f)) %>%
  ggplot(data=., aes(x=mage, y=wksgest,
                     color=pnc5_f)) +
  geom_point(alpha=0.5) +
  facet_grid(cigdur_f~raceeth_f) +
  ggtitle("Facet by 2 Discrete Variables")
```



# Check-in


YOU KNOW THIS METAL RECTANGLE FULL OF LITTLE LIGHTS?
YEAH.
I SPEND MOST OF MY LIFE PRESSING BUTTONS TO MAKE THE PATTERN OF LIGHTS CHANGE HOWEVER I WANT.
SOUNDS GOOD.
BUT TODAY, THE PATTERN OF LIGHTS IS ALL WRONG!
OH GOD! TRY PRESSING MORE BUTTONS!
IT'S NOT HELPING!

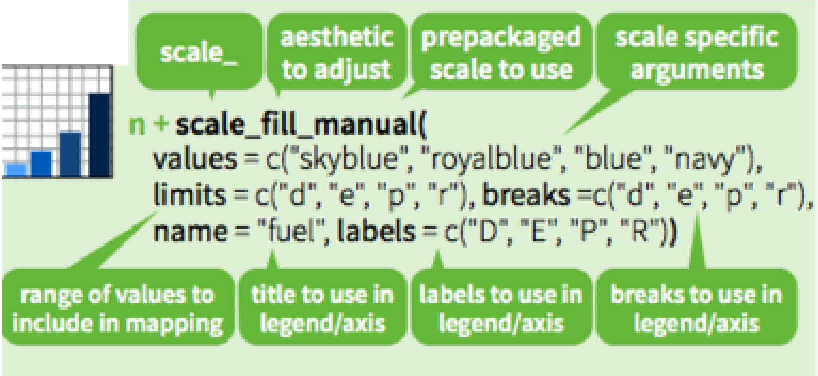
# Additonal Important Features of ggplot

**Today**

- *Coordinate systems* if don't want to use default cartesian coordinates.
- *Facets* to divide a plot into subplots based on the values of one or more discrete variables.
- *Scales* to adjust aesthetics and colors (`scale_*_*()`)
- *Themes* to modify the overall appearance of the plot (background, grid lines).

# Scales (1)

"Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale". Basically, the first "*" is the aesthetic we want to adjust and the second "*" is scale argument/function that we are calling (pre-built) in `scale_*_*()`. Or – said another way – "Note the naming scheme for scales: scale_ followed by the name of the aesthetic, then _, then the name of the scale."



```
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"))
```

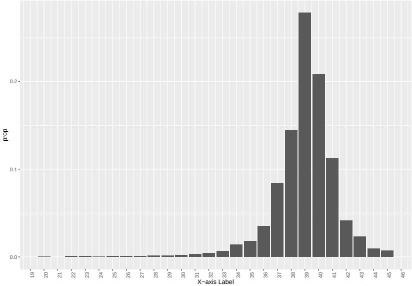
scale_
aesthetic to adjust
prepackaged scale to use
scale specific arguments
range of values to include in mapping
title to use in legend/axis
labels to use in legend/axis
breaks to use in legend/axis

# Scales (2)

“The default scales are named according to the type of variable they with: continuous, discrete, datetime, or date.”

Typically we use these scales to change the breaks or interpretation of our x and y aesthetics.

```
ggplot(data=births,
       aes(x=wksgest, y=..prop..)) +
  geom_bar() +
  scale_x_continuous(name="X-axis Label",
                     breaks=seq(from=10,
                                to=60, by=1)) +
  theme(axis.text.x = element_text(angle=90))
```

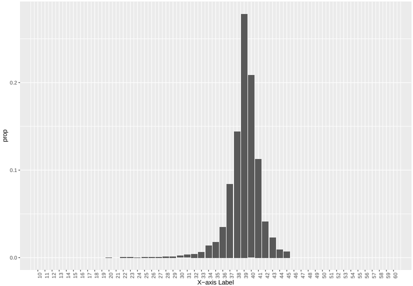


Thanks, R for Data Science

# Scales (3)

By default, our limits min and max is the min and max of the data (+/-1). We can coerce R to overextend our plot with the `limits` argument.

```
ggplot(data=births,
       aes(x=wksgest, y=..prop..)) +
  geom_bar() +
  scale_x_continuous(name="X-axis Label",
                     breaks=seq(from=10,
                                to=60, by=1),
                     limits=c(10,60)) +
  theme(axis.text.x = element_text(angle=90))
```

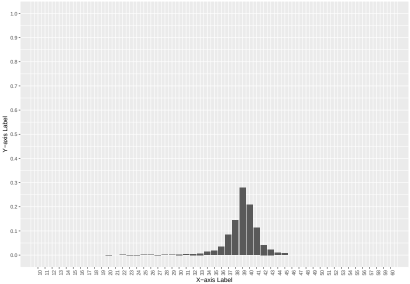


Thanks, R for Data Science

# Scales (4)

We can do the same thing on the y-axis.

```
ggplot(data=births,
       aes(x=wksgest, y=..prop..)) +
  geom_bar() +
  scale_x_continuous(name="X-axis Label",
                     breaks=seq(from=10,
                                to=60, by=1),
                     limits=c(10,60)) +
  scale_y_continuous(name="Y-axis Label",
                     breaks=seq(from=0,
                                to=1, by=0.1),
                     limits=c(0,1)) +
  theme(axis.text.x = element_text(angle=90))
```

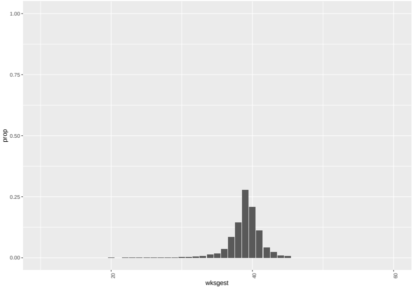


# Scales (5)

## Aside

If you just want to change the x- or y-axis limits, and not change the scale of the x- or y-axis, just consider using `xlim` and/or `ylim`.

```
ggplot(data=births,
       aes(x=wksgest, y=..prop..)) +
  geom_bar() +
  xlim(10,60) +
  ylim(0,1) +
  theme(axis.text.x = element_text(angle=90))
```

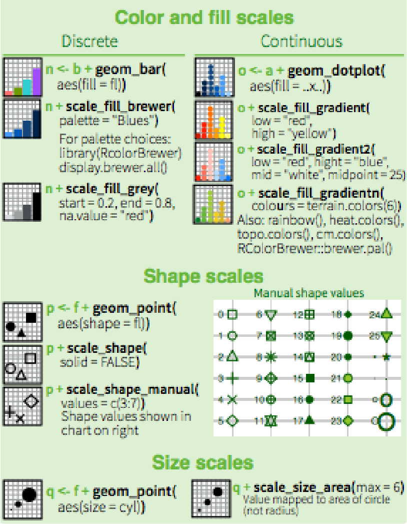


# Scale (6)

Perhaps, more importantly, scales can be used for

- Color
- Shape
- Size

aesthetic manipulations.



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))
```

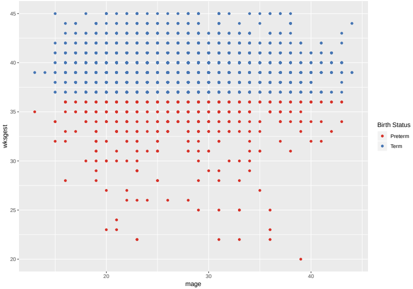
```
q + scale_size_area(max = 6)
```

Value mapped to area of circle (not radius)

Thanks, ggplot-cheatsheet

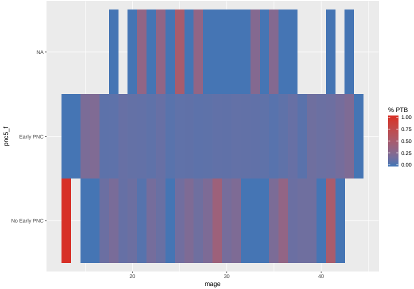
# Scale (7)

```r
ggplot(data=births,
       aes(x=mage, y=wksgest,
           color=preterm_f)) +
  geom_point() +
  scale_color_manual(name="Birth Status",
                     labels =
                       c("Preterm", "Term"),
                     values =
                       c("#d73027", "#4575b4"))
```



# Scale (7)

```
births %>%
  group_by(mage, pnc5_f) %>%
  summarise(meanPTB = mean(preterm)) %>%
ggplot(data=.,
       aes(x=mage, y=pnc5_f,
           fill=meanPTB)) +
  geom_tile() +
  scale_fill_gradient(name="% PTB",
                      low = "#d73027",
                      high = "#4575b4")
```

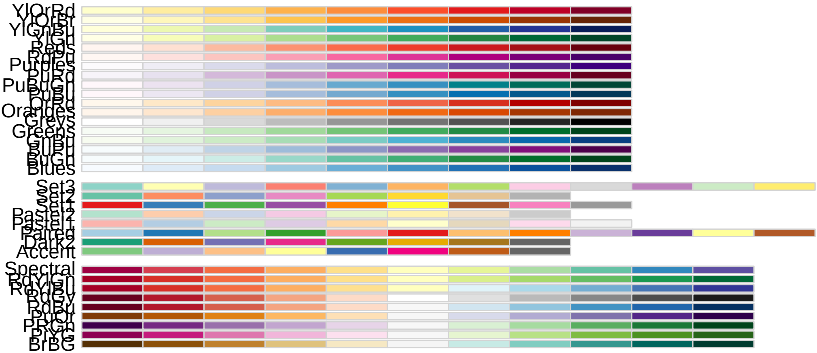


# A Note On Color Selection (1)

Data visualization and the proper communication of data with colors, shapes, labels, etc. I highly recommend this site: colorbrewer.org and the R-package `RColorBrewer`. `RColorBrewer` come pre-installed with color palettes that are either qualitative, diverging, or sequential (the website is an extension of the package).

# A Note On Color Selection (2)

```r
library(RColorBrewer)
RColorBrewer::display.brewer.all()
```



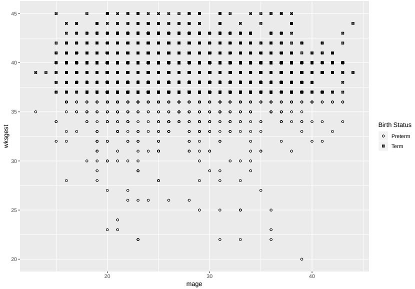
# A Note On Color Selection (3)

ggsci offers a collection of ggplot2 color palettes inspired by scientific journals, data visualization libraries, science fiction movies, and TV shows

Thanks, Mike!

# Scale (8)

```r
ggplot(data=births,
       aes(x=mage, y=wksgest,
           shape=preterm_f)) +
  geom_point() +
  scale_shape_manual(name="Birth Status",
                     labels =
                       c("Preterm", "Term"),
                     values =
                       c(1,7))
```



# Themes

There are two major flavors of themes:

1. pre-built themes
2. customized/self-produced themes

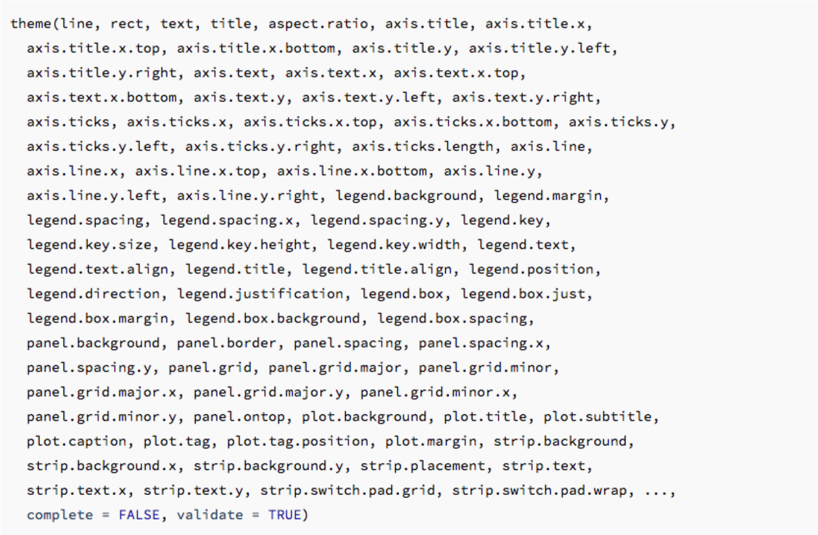
All themes are appendable, so you can change the “settings” in any pre-built theme.

# Pre-built themes

These are appendable objects on your ggplot object and include:

- `theme_bw()`
- `theme_classic()`
- `theme_minimal()`
- `theme_light()`
- Even more with ggthemes

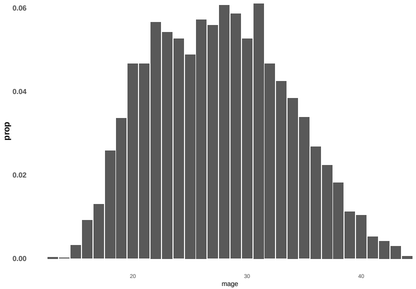
# Customized Themes (1)



```
theme(line, rect, text, title, aspect.ratio, axis.title, axis.title.x,
  axis.title.x.top, axis.title.x.bottom, axis.title.y, axis.title.y.left,
  axis.title.y.right, axis.text, axis.text.x, axis.text.x.top,
  axis.text.x.bottom, axis.text.y, axis.text.y.left, axis.text.y.right,
  axis.ticks, axis.ticks.x, axis.ticks.x.top, axis.ticks.x.bottom, axis.ticks.y,
  axis.ticks.y.left, axis.ticks.y.right, axis.ticks.length, axis.line,
  axis.line.x, axis.line.x.top, axis.line.x.bottom, axis.line.y,
  axis.line.y.left, axis.line.y.right, legend.background, legend.margin,
  legend.spacing, legend.spacing.x, legend.spacing.y, legend.key,
  legend.key.size, legend.key.height, legend.key.width, legend.text,
  legend.text.align, legend.title, legend.title.align, legend.position,
  legend.direction, legend.justification, legend.box, legend.box.just,
  legend.box.margin, legend.box.background, legend.box.spacing,
  panel.background, panel.border, panel.spacing, panel.spacing.x,
  panel.spacing.y, panel.grid, panel.grid.major, panel.grid.minor,
  panel.grid.major.x, panel.grid.major.y, panel.grid.minor.x,
  panel.grid.minor.y, panel.ontop, plot.background, plot.title, plot.subtitle,
  plot.caption, plot.tag, plot.tag.position, plot.margin, strip.background,
  strip.background.x, strip.background.y, strip.placement, strip.text,
  strip.text.x, strip.text.y, strip.switch.pad.grid, strip.switch.pad.wrap, ...,
  complete = FALSE, validate = TRUE)
```

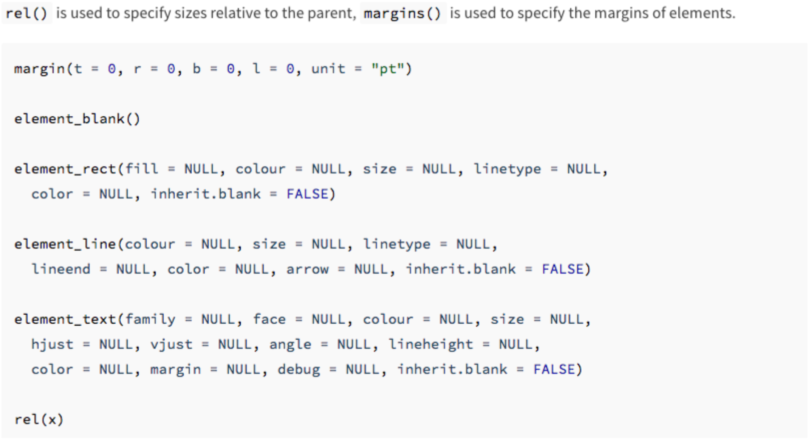
# Customized Themes (2)

```
ggplot(data=births,
       aes(x=mage, y=..prop..)) +
  geom_bar() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.ticks = element_blank(),
        axis.text.x = element_text(size=9),
        axis.title.y = element_text(size=14,
                                    face="bold"),
        axis.text.y = element_text(size=12,
                                   face="bold")
          )
```



# Theme Elements

"... the element_ functions specify the display of how non-data components of the plot are a drawn" as part of the theme argument. Basically, `element_blank` assigns nothing to that property, `element_rect` is for borders and backgrounds, `element_line` and `element_text` are for lines and text, respectively.



`rel()` is used to specify sizes relative to the parent, `margins()` is used to specify the margins of elements.

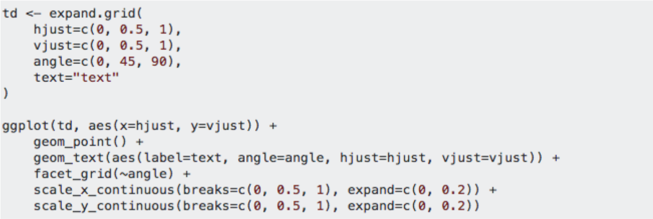
```
margin(t = 0, r = 0, b = 0, l = 0, unit = "pt")

element_blank()

element_rect(fill = NULL, colour = NULL, size = NULL, linetype = NULL,
  color = NULL, inherit.blank = FALSE)

element_line(colour = NULL, size = NULL, linetype = NULL,
  lineend = NULL, color = NULL, arrow = NULL, inherit.blank = FALSE)

element_text(family = NULL, face = NULL, colour = NULL, size = NULL,
  hjust = NULL, vjust = NULL, angle = NULL, lineheight = NULL,
  color = NULL, margin = NULL, debug = NULL, inherit.blank = FALSE)

rel(x)
```

# Theme Elements in practice (1)

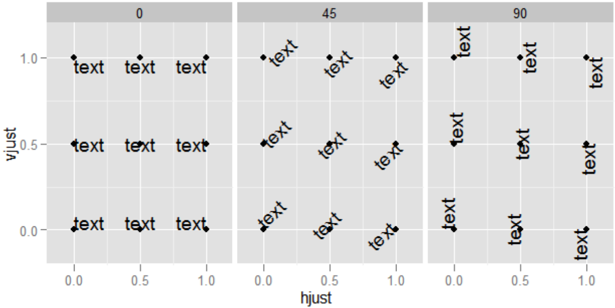
```r
ggplot(data=births, aes(x=mage, y=wksgest)) +
  geom_point() +
  labs(title = "Title",
       subtitle = "Subtitle",
       caption = "caption",
       x = "x",
       y = "y") +
    theme(plot.title = element_text(hjust = 0.5, family="Arial", size=17, face = "bold"),
          plot.subtitle = element_text(hjust = 0, family="Arial", size=15, face = "italic"),
          plot.caption = element_text(hjust = 1, family="Arial", size=12, face = "italic"),
          axis.text.x = element_text(family="Arial", size=13, face = "bold", angle=90, vjust=0.5),
          axis.title.x = element_text(family="Arial", size=15, face = "bold"),
          axis.text.y = element_text(family="Arial", size=13, face = "bold", hjust=0.5),
          axis.title.y = element_text(family="Arial", size=15, face = "bold"),
          legend.title = element_text(family="Arial", size=12, face = "bold"),
          legend.text = element_text(family="Arial", size=10, face = "bold"),
          legend.title.align = 0.5,
          panel.background = element_blank(),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          axis.line = element_line(colour = "black", size=2),
          axis.ticks = element_blank()
          )
```

# Theme Elements in practice (2)

## Vertical and Horizontal Adjustments



```
td <- expand.grid(
    hjust=c(0, 0.5, 1),
    vjust=c(0, 0.5, 1),
    angle=c(0, 45, 90),
    text="text"
)

ggplot(td, aes(x=hjust, y=vjust)) +
    geom_point() +
    geom_text(aes(label=text, angle=angle, hjust=hjust, vjust=vjust)) +
    facet_grid(~angle) +
    scale_x_continuous(breaks=c(0, 0.5, 1), expand=c(0, 0.2)) +
    scale_y_continuous(breaks=c(0, 0.5, 1), expand=c(0, 0.2))
```

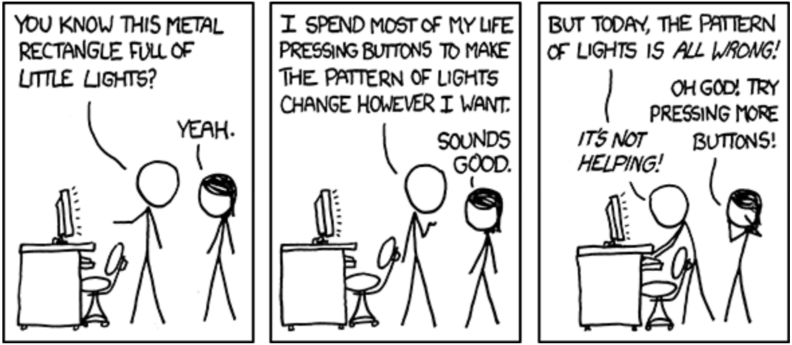


Thank you, Andrie from StackExchange

# Themes

Options are endless. Feel free to email me if you have specific questions but the best way to learn is to play around with the code/options. I recommend making your own theme and using that as your own personal brand. I've had friends receive reviewer comments that their plot “looks like it came straigh from ggplot”. So take this opportunity to show your individuality/style!

# Check-in

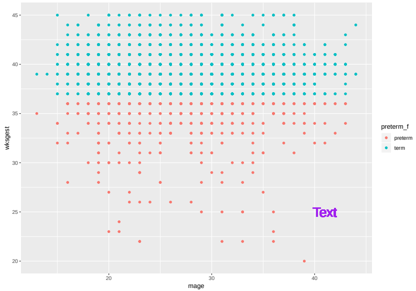

YOU KNOW THIS METAL RECTANGLE FULL OF LITTLE LIGHTS?
YEAH.
I SPEND MOST OF MY LIFE PRESSING BUTTONS TO MAKE THE PATTERN OF LIGHTS CHANGE HOWEVER I WANT.
SOUNDS GOOD.
BUT TODAY, THE PATTERN OF LIGHTS IS ALL WRONG!
OH GOD! TRY PRESSING MORE BUTTONS!
IT'S NOT HELPING!

# Addition Geoms To Consider (1)

- Predefined interval geom_ribbon (on top of geom_lines)
- geom_text, geom_label (see ggrepel for better use-cases)

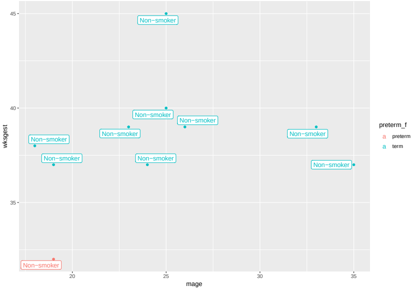
# geom_text

```
ggplot(data=births,
       aes(x=mage, y=wksgest,
           color=preterm_f)) +
  geom_point() +
  geom_text(aes(x=41, y=25, label="Text"),
          color="purple")
```



# geom_label

```
library(ggrepel)
births[sample(1:nrow(births), 10, replace=F), ] %>%
  mutate(cigdur_f = factor(cigdur,
                           levels=c(0,1),
                           labels=c("Non-smoker",
                                    "Smoker"))) %>%
  ggplot(data=.,
      aes(x=mage, y=wksgest,
          color=preterm_f)) +
  geom_point() +
  ggrepel::geom_label_repel(aes(label=cigdur_f))
```



# Multiple Plots on a Single Page (1)

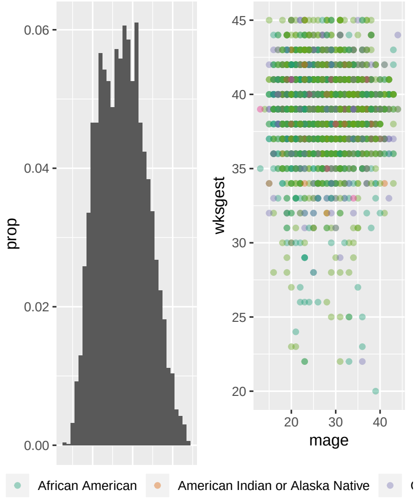
## Set-up

```
plotobj1 <- ggplot() +
  geom_bar(data=births, aes(x=mage, y=..prop..))

plotobj2 <- ggplot() +
  geom_point(data=births,
             aes(x=mage, y=wksgest, color=raceeth_f),
             alpha=0.4) +
  scale_color_brewer(palette = "Dark2") +
  theme(legend.position = "bottom")

# could call these with plot like this (if we wanted)
# plot(plotobj1)
# plot(plotobj2)
```

# Multiple Plots on a Single Page (2)

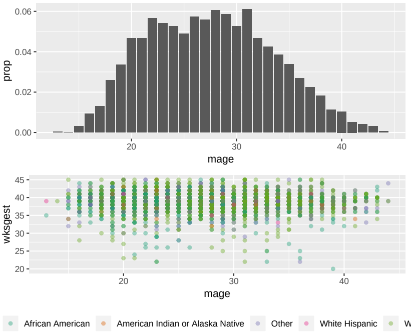
Here we are going to use the `gridExtra` package to put two plots on a single page (and we can format this however we want).

```r
library(gridExtra)

gridExtra::grid.arrange(plotobj1, plotobj2, ncol=2)
```



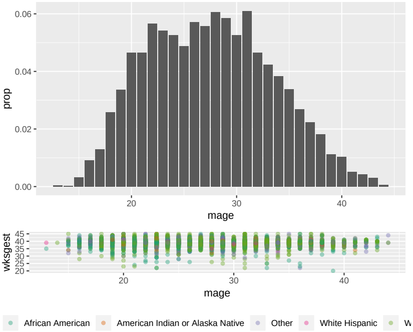
# Multiple Plots on a Single Page (3)

```r
gridExtra::grid.arrange(plotobj1, plotobj2, nrow=2)
```



# Multiple Plots on a Single Page (4)

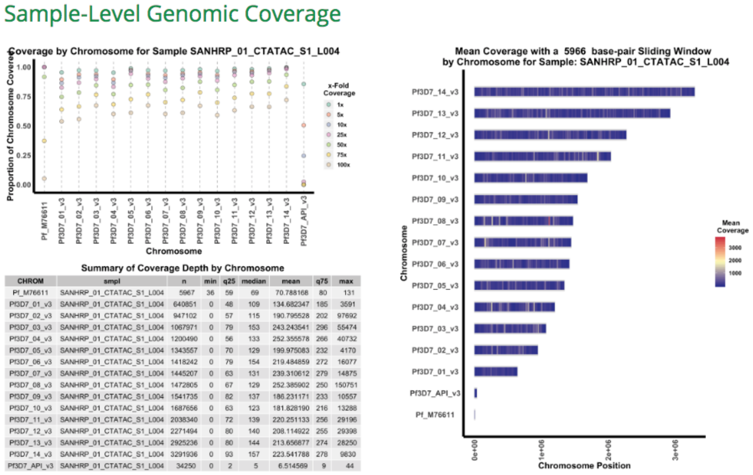
```
gridExtra::grid.arrange(plotobj1, plotobj2,
                        layout_matrix =
                          rbind(1,1,2))

rbind(1,1,2)
# call this on your own to understand
```



# Multiple Plots on a Single Page (5)

`GridExtra` can be much, much more complex and advanced (again, it will let you do anything)! It is a great package. It will even let you plot tables and graphs side-by-side or scaled however you would like. Definitely worth the time-investment.

# Personal Example with `GridExtra`



## Sample-Level Genomic Coverage

Summary of Coverage Depth by Chromosome

| CHROM | smpl | n | min | q25 | median | mean | q75 | max |
|---|---|---|---|---|---|---|---|---|
| Pf_M76611 | SANHRP_01_CTATAC_S1_L004 | 5967 | 36 | 59 | 69 | 70.788168 | 80 | 131 |
| Pf3D7_01_v3 | SANHRP_01_CTATAC_S1_L004 | 640851 | 0 | 48 | 109 | 134.682347 | 185 | 3591 |
| Pf3D7_02_v3 | SANHRP_01_CTATAC_S1_L004 | 947102 | 0 | 57 | 115 | 190.795528 | 202 | 97692 |
| Pf3D7_03_v3 | SANHRP_01_CTATAC_S1_L004 | 1067971 | 0 | 79 | 153 | 243.243541 | 296 | 55474 |
| Pf3D7_04_v3 | SANHRP_01_CTATAC_S1_L004 | 1200490 | 0 | 56 | 133 | 252.355578 | 266 | 40732 |
| Pf3D7_05_v3 | SANHRP_01_CTATAC_S1_L004 | 1343557 | 0 | 70 | 129 | 199.975083 | 232 | 4170 |
| Pf3D7_06_v3 | SANHRP_01_CTATAC_S1_L004 | 1418242 | 0 | 79 | 154 | 219.484859 | 272 | 16077 |
| Pf3D7_07_v3 | SANHRP_01_CTATAC_S1_L004 | 1445207 | 0 | 63 | 131 | 239.310612 | 279 | 14875 |
| Pf3D7_08_v3 | SANHRP_01_CTATAC_S1_L004 | 1472805 | 0 | 67 | 129 | 252.385902 | 250 | 150751 |
| Pf3D7_09_v3 | SANHRP_01_CTATAC_S1_L004 | 1541735 | 0 | 82 | 137 | 186.231171 | 233 | 10557 |
| Pf3D7_10_v3 | SANHRP_01_CTATAC_S1_L004 | 1687656 | 0 | 63 | 123 | 181.828190 | 216 | 13288 |
| Pf3D7_11_v3 | SANHRP_01_CTATAC_S1_L004 | 2038340 | 0 | 72 | 139 | 220.251133 | 256 | 29196 |
| Pf3D7_12_v3 | SANHRP_01_CTATAC_S1_L004 | 2271494 | 0 | 80 | 140 | 208.114922 | 255 | 29398 |
| Pf3D7_13_v3 | SANHRP_01_CTATAC_S1_L004 | 2925236 | 0 | 80 | 144 | 213.656877 | 274 | 28250 |
| Pf3D7_14_v3 | SANHRP_01_CTATAC_S1_L004 | 3291936 | 0 | 93 | 157 | 223.541788 | 278 | 9830 |
| Pf3D7_API_v3 | SANHRP_01_CTATAC_S1_L004 | 34250 | 0 | 2 | 5 | 6.514569 | 9 | 44 |

# Interactive ggplot using plotly (1)

```r
library(plotly)
plotObj <- ggplot(data=births,
                  aes(x=mage, y=wksgest, color=preterm_f)) +
  geom_point()
plotly::ggplotly(plotObj)
```

# Interactive ggplot using plotly (2)

## Hacks

You can put non-aesthetics into the aesthetic calls, which will get picked up by the plotly labels.

```r
library(plotly)
plotObj <- ggplot(data=births, aes(x=mage, y=wksgest,
                                   color=preterm_f,
                                   cigdur=cigdur,
                                   pnc5_f=pnc5_f)) +
  geom_point()
plotly::ggplotly(plotObj)
```

Thanks for the hack, Nick Hathaway

# Interactive ggplot using `plotly` (3)

`plotly` has its own graphics syntax (and functions). Sometimes they are very useful (advanced content), but I usually default to the `plotly::ggplotly(plotObj)` minimum use-case

# Saving Plots

## ggplot way

```
ggsave(filename=<>, device="png",
       height=8, width=11, units="in", res=500)
# defaults to last plot you made
```

## base way

```
jpeg(filename = <>, height=8,
     width=11, units="in", res=500) # could also use svg or png
plot(<plotobj>)
graphics.off()
```

# ggplot Extensions

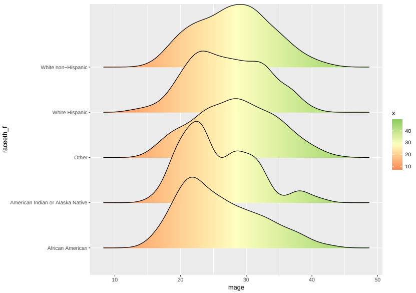
This website contains ggplot extentsions in the language we now know: http://www.ggplot2-exts.org/

Other items of interest (overlapping):

- Predefined interval geom_ribbon (on top of geom_lines)
- Heatmaps (geom_tile or geom_rect)
- gganimations
- DAGs
- PCA
- Anatomy (below)
- Networks (below)
- Chromosome painting
- Dendograms
- Snakey

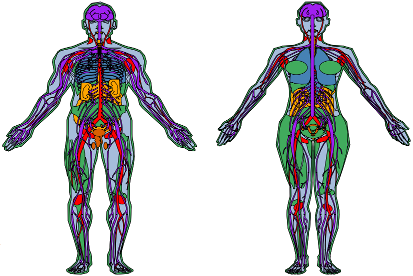
# ggridges

```
library(ggridges)
ggplot() +
  geom_density_ridges_gradient(data=births,
                               aes(x=mage,
                                   y=raceeth_f,
                                   fill=..x..)) +
  scale_fill_gradientn(colors=c("#fc8d59",
                                "#ffffbf", "#91cf60"))
```



# gganatogram

```r
# library(devtools)
devtools::install_github("jespermaag/gganatogram")
library(gganatogram)
hgMale <- gganatogram(data=hgMale_key,
                      fillOutline='#a6bddb',
                      organism='human',
                      sex='male', fill="colour") +
  theme_void()
hgFemale <- gganatogram(data=hgFemale_key,
                        fillOutline='#a6bddb',
                        organism='human',
                        sex='female', fill="colour") +
  theme_void()

gridExtra::grid.arrange(hgMale, hgFemale, ncol=2)
```

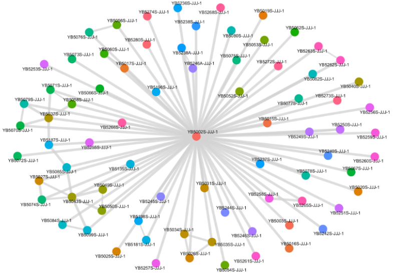


Thank you, Maag J

# Networks with tidygraph

Networks typically depend on a distance-metric (i.e. edges)

```
# library(devtools)
devtools::install_github("thomasp85/tidygraph")
library(tidygraph)
library(ggraph)
# dab is a distance
ggraph(<data>) +
  geom_edge_fan(aes(width=<distance>),
                colour = "#d9d9d9", alpha=0.8) +
  scale_edge_width(range = c(0.8, 1.5)) +
  geom_node_point(aes(color = name, size=2.5)) +
  geom_node_text(aes(label = name),
                 size = 2, repel = TRUE) +
  theme_graph() +
  theme(legend.position = "none")
```


YB5236S-JJJ-1
YB5274S-JJJ-1
YB5268S-JJJ-1
YB5019S-JJJ-1
YB5056S-JJJ-1
YB5238S-JJJ-1
YB5080S-JJJ-1
YB5076S-JJJ-1
YB5062S-JJJ-1
YB5280S-JJJ-1
YB5053S-JJJ-1
YB5060S-JJJ-1
YB5238A-JJJ-1
YB5263S-JJJ-1
YB5073S-JJJ-1
YB5075S-JJJ-1
YB5272S-JJJ-1
YB5253S-JJJ-1
YB5017S-JJJ-1
YB5246A-JJJ-1
YB5262S-JJJ-1
YB5082S-JJJ-1
YB5071S-JJJ-1
YB5196S-JJJ-1
YB5066S-JJJ-1
YB5052S-JJJ-1
YB5040S-JJJ-1
YB5079S-JJJ-1
YB5058S-JJJ-1
YB5273S-JJJ-1
YB5077S-JJJ-1
YB5037S-JJJ-1
YB5256S-JJJ-1
YB5015S-JJJ-1
YB5266S-JJJ-1
YB5070S-JJJ-1
YB5002S-JJJ-1
YB5250S-JJJ-1
YB5249S-JJJ-1
YB5259S-JJJ-1
YB5187S-JJJ-1
YB5255S-JJJ-1
YB5240S-JJJ-1
YB5072S-JJJ-1
YB5237S-JJJ-1
YB5260S-JJJ-1
YB5135S-JJJ-1
YB5067S-JJJ-1
YB5085S-JJJ-1
YB5078S-JJJ-1
YB5027S-JJJ-1
YB5031S-JJJ-1
YB5049S-JJJ-1
YB5020S-JJJ-1
YB5258S-JJJ-1
YB5245S-JJJ-1
YB5043S-JJJ-1
YB5074S-JJJ-1
YB5050S-JJJ-1
YB5265S-JJJ-1
YB5244S-JJJ-1
YB5251S-JJJ-1
YB5198S-JJJ-1
YB5084S-JJJ-1
YB5003S-JJJ-1
YB5099S-JJJ-1
YB5034S-JJJ-1
YB5246S-JJJ-1
YB5242S-JJJ-1
YB5181S-JJJ-1
YB5035S-JJJ-1
YB5016S-JJJ-1
YB5025S-JJJ-1
YB5026S-JJJ-1
YB5261S-JJJ-1
YB5257S-JJJ-1
YB5054S-JJJ-1