Lecture 09

Graphics::ggplot I

2018 R Teaching Team

October 1, 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. Garrett Grolemund for `gganimate`
5. Garrick Aden-Buie for tidyverse animations
6. Hadley for R for Data Scientists and Advanced R
7. R & Rstudio Teams

This Week

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

This Lecture

The Tidyverse

A collection of packages for data manipulation, exploration, and visualization.

Share a common philosophy of R programming and work in harmony.

Core tidyverse packages:

`dplyr` `ggplot2` `purrr` `tibble` `tidyr` `readr`

This Week's Focus

Thanks, Sara!

This Lecture

The Tidyverse

A collection of packages for data manipulation, exploration, and visualization.

Share a common philosophy of R programming and work in harmony.

Core tidyverse packages:

`dplyr` `ggplot2` `purrr` `tibble` `tidyr` `readr`

Let's Review/Introduce these packages first!

Thanks, Sara!

This Lecture

The Tidyverse

A collection of packages for data manipulation, exploration, and visualization.

Share a common philosophy of R programming and work in harmony.

Core tidyverse packages:

`dplyr` `ggplot2` `purrr` `tibble` `tidyr` `readr`

And we will return to this later

Thanks, Sara!

Overview of Lecture

TIDYVERSE & GGPLOT!!

Structure

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

Clarification Point

The %>% function is from the `magrittr` package. From the `magrittr` website:

Abstract

The magrittr (to be pronounced with a sophisticated french accent) is a package with two aims: to decrease development time and to improve readability and maintainability of code. Or even shortr: to make your code smokin' (puff puff)!

To archive its humble aims, *magrittr* (remember the accent) provides a new "pipe"-like operator, `%>%`, with which you may pipe a value forward into an expression or function call; something along the lines of `x %>% f`, rather than `f(x)`. This is not an unknown feature elsewhere; a prime example is the `|>` operator used extensively in `F#` (to say the least) and indeed this – along with Unix pipes – servec as a motivation for developing the magrittr package.

This vignette describes the main features of *magrittr* and demonstrates some features which has been added since the initial release.

Clarification Point

This %>% function can be used with any other functions, f(x) (not just `tidyverse` functions). Of note, the "." is a placeholder (more explicit code/can be useful for cases where it is not "obvious" where x should go in f(x)).

```
"hi" %>% paste %>% toupper
```

```
## [1] "HI"
```

```
"hi" %>% paste(.) %>% toupper(.)
```

```
## [1] "HI"
```

What does it mean to be “tidy”

Thank you, R for Data Scientists

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.

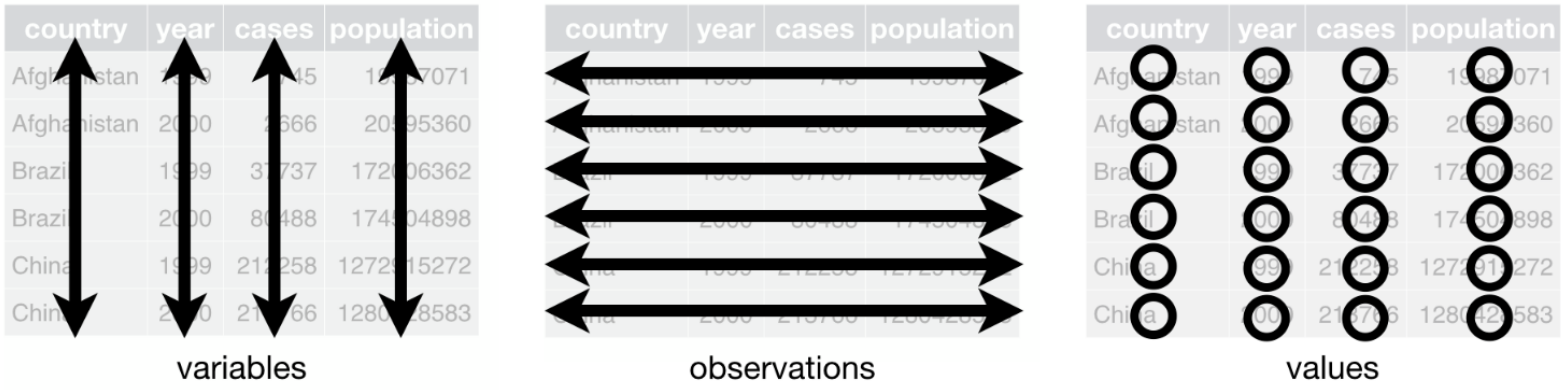


Figure 12.1: Following three rules makes a dataset tidy: variables are in columns, observations are in rows, and values are in cells.

What does it mean to be “tidy”

To some, this concept may seem “obvious” but is really powerful!
For a great read, see Hadley Wickham’s paper on tidy data here.

Is our Births data set set up in a “tidy” format?

Cool, let's read it in!

Note, using `readr::read_csv` here!

```
FALSE [1] "~/Documents/GitHub/18Fall_EPID799C_RforEpi/data/
```

```
births <- readr::read_csv(file="<path>",
                          col_names = T)

# ?read.csv
```

Note

I have made my dataset much smaller by randomly sampling 5,000 observations (so the presentation can fit on the website). You should not do this for your homework, etc.

What's the big difference?

```
births[, c("preterm", "pnc5", "mage")]
```

```
## # A tibble: 5,000 x 3
##    preterm  pnc5  mage
##      <int> <int> <int>
##  1       0     1    32
##  2       0     1    22
##  3       0    NA    24
##  4       0     1    32
##  5       0     1    22
##  6       0     1    29
##  7       0     1    20
##  8       0     1    29
##  9       0     1    27
## 10       0     1    25
## # ... with 4,990 more rows
```

What is a Tibble

From R for Data Scientists: "Tibbles are data frames, but they tweak some older behaviours to make life a little easier. [For example, tibbles will] never changes the type of the inputs (e.g. it never converts strings to factors!), it never changes the names of variables, and it never creates row names. [Plus it can] have column names that are not valid R variable names, aka **non-syntactic names** [`@Newcolumnname`]"

What is a Tibble

More from R for Data Scientists: There are two main differences in the usage of a tibble vs. a classic data.frame: printing and subsetting.

Printing: "Tibbles have a refined print method that shows only the first 10 rows, and all the columns that fit on screen."

Subsetting: To subset, "[[can extract by name or position; $ only extracts by name but is a little less typing." Basically, "compared to a data.frame, tibbles are more strict: they never do partial matching, and they will generate a warning if the column you are trying to access does not exist."

In subsetting the main difference between data.frames and working "with tibble is the [function. We don't use [much in this book because dplyr::filter() and dplyr::select() allow you to solve the same problems with clearer code (but you will learn a little about it in vector subsetting). With base R data frames, [sometimes returns a data frame, and sometimes returns a vector. With tibbles, [always returns another tibble."

What if our data was not “tidy”?

Thank you, R for Data Scientists

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.

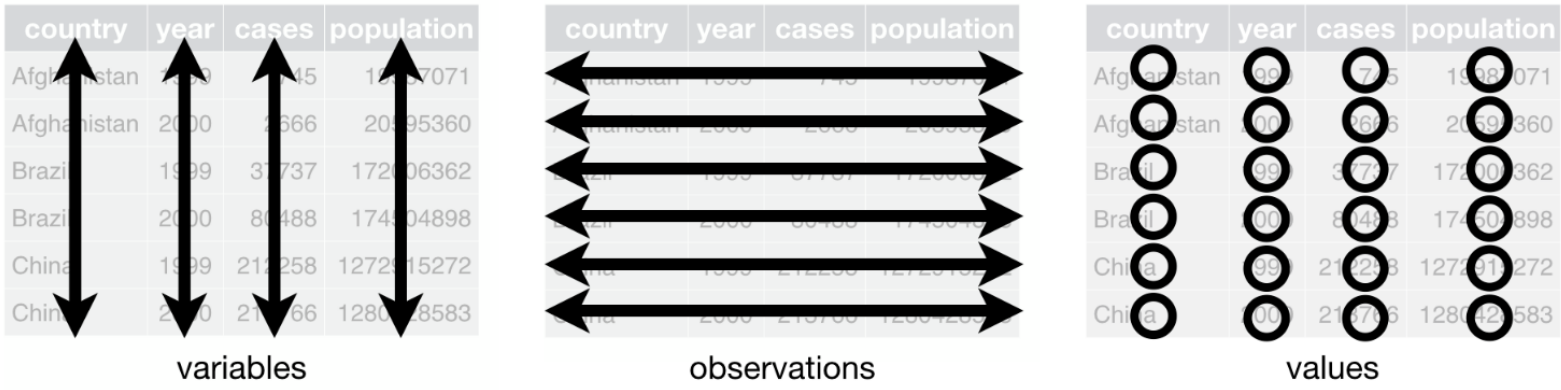
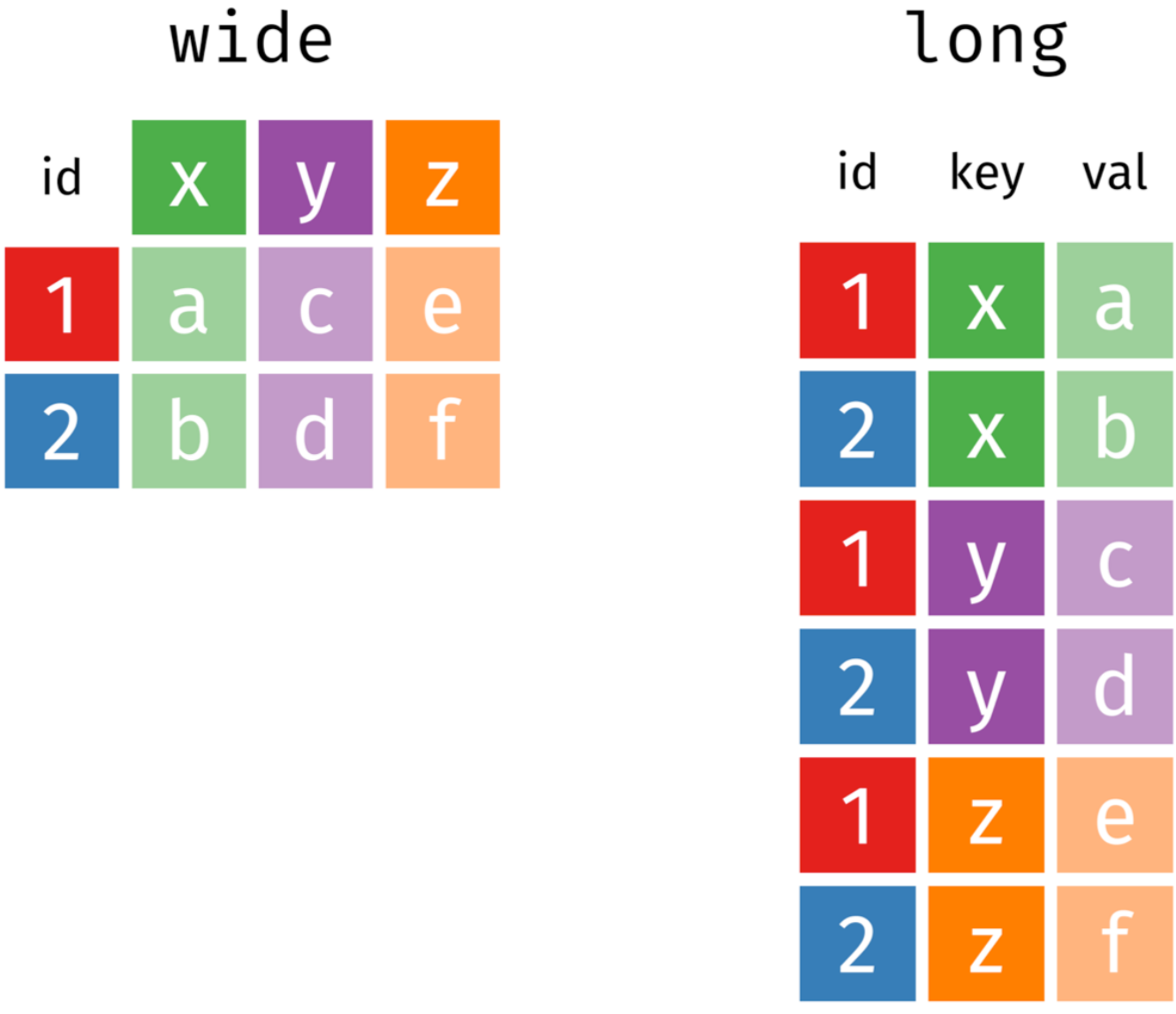


Figure 12.1: Following three rules makes a dataset tidy: variables are in columns, observations are in rows, and values are in cells.

Spread and Gather

Thank you, Garrick Aden-Buie!



Spread and Gather

GIF here

Thank you, Garrick Aden-Buie!

Spread and Gather in Real Epi Life Example (genetics)

Let's say that you had a variant call file, which contains loci (i.e. Chromosome, genomic location, reference allele, alternative allele) as rows and sample IDs as columns. In the simplest case, under the sample ID headers are genotypes (either 1 or 0s) that say the person has this SNP or not. Let's take a peak.

```
vcf
```

```
## # A tibble: 55 x 10
##    CHROM POS   REF   ALT   `7G8` ERS740936 ERS740937 ERS740940 GB4
##    <chr> <chr> <chr> <chr> <chr> <chr>     <chr>     <chr>     <chr>
##  1 Pf3D~ 95632 G     A     0     0         0         0         0
##  2 Pf3D~ 95641 G     A     0     0         0         0         0
##  3 Pf3D~ 95680 A     G     0     0         0         0         0
##  4 Pf3D~ 95685 G     T     0     0         0         0         0
##  5 Pf3D~ 95686 A     C     0     0         0         0         0
##  6 Pf3D~ 1003~ G     C     0     0         1         0         0
##  7 Pf3D~ 1003~ A     G     0     1         1         0         1
##  8 Pf3D~ 1003~ G     A     0     0         0         0         0
##  9 Pf3D~ 1012~ G     T     1     1         1         <NA>      1
## 10 Pf3D~ 1031~ G     A     0     0         0         0         0
## # ... with 45 more rows, and 1 more variable: `PA0007-C` <chr>
```

Making VCF tidy (i.e. "long" format)

```
longvcf <- vcf %>%
  tidyr::gather(data=., key="ID", value="GT", 5:ncol(vcf)) %>%
  dplyr::arrange(CHROM, POS)

longvcf
```

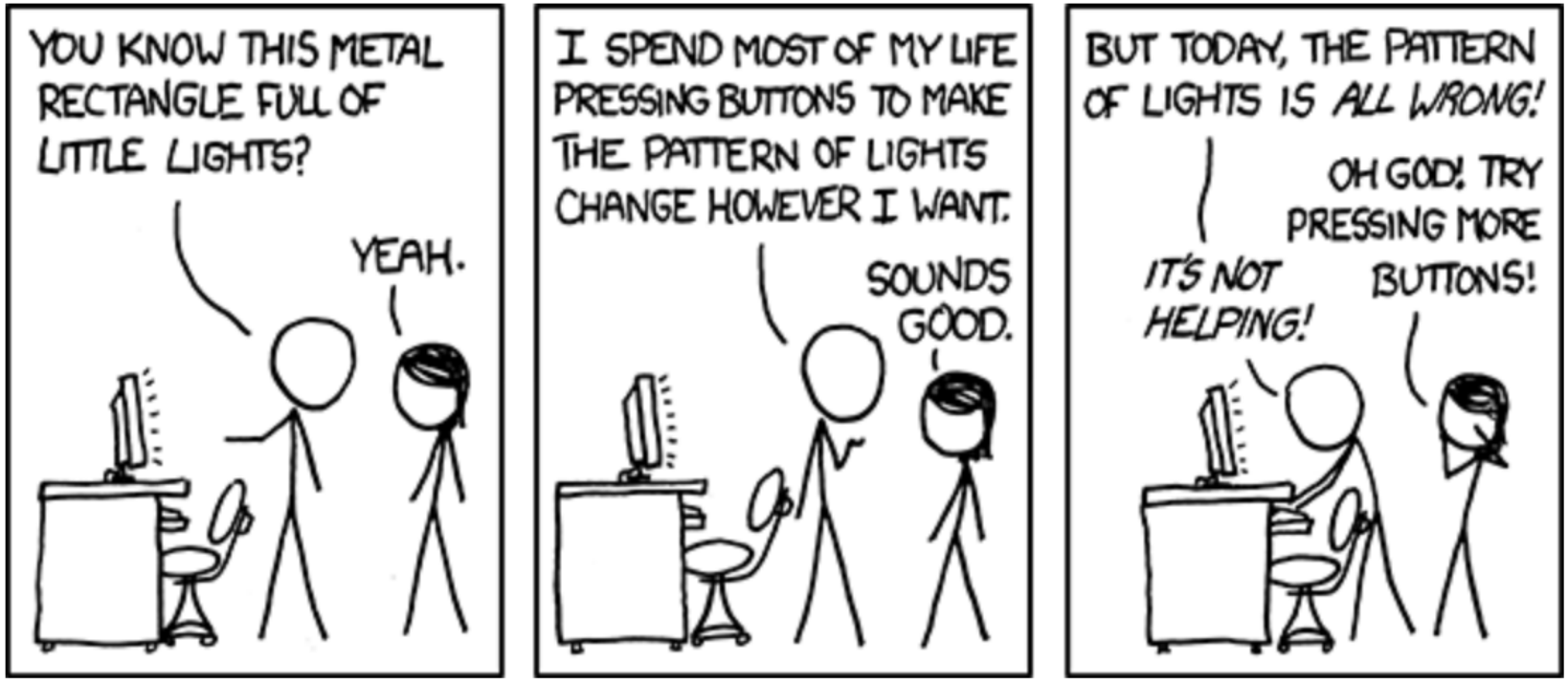
```
## # A tibble: 330 x 6
##    CHROM       POS    REF   ALT   ID         GT
##    <chr>       <chr>  <chr> <chr> <chr>      <chr>
##  1 Pf3D7_01_v3 100315 G     C     7G8        0
##  2 Pf3D7_01_v3 100315 G     C     ERS740936  0
##  3 Pf3D7_01_v3 100315 G     C     ERS740937  1
##  4 Pf3D7_01_v3 100315 G     C     ERS740940  0
##  5 Pf3D7_01_v3 100315 G     C     GB4        0
##  6 Pf3D7_01_v3 100315 G     C     PA0007-C   0
##  7 Pf3D7_01_v3 100330 A     G     7G8        0
##  8 Pf3D7_01_v3 100330 A     G     ERS740936  1
##  9 Pf3D7_01_v3 100330 A     G     ERS740937  1
## 10 Pf3D7_01_v3 100330 A     G     ERS740940  0
## # ... with 320 more rows
```

What if we wanted wide format?

```
longvcf %>%
  tidyr::spread(data=., key="ID", value="GT")
```

```
## # A tibble: 55 x 10
##    CHROM POS   REF   ALT   `7G8` ERS740936 ERS740937 ERS740940 GB4
##    <chr> <chr> <chr> <chr> <chr> <chr>     <chr>     <chr>     <chr>
##  1 Pf3D~ 1003~ G     C     0     0         1         0         0
##  2 Pf3D~ 1003~ A     G     0     1         1         0         1
##  3 Pf3D~ 1003~ G     A     0     0         0         0         0
##  4 Pf3D~ 1012~ G     T     1     1         1         <NA>      1
##  5 Pf3D~ 1031~ G     A     0     0         0         0         0
##  6 Pf3D~ 1066~ G     A     0     0         0         0         0
##  7 Pf3D~ 1085~ G     A     0     1         0         0         0
##  8 Pf3D~ 1110~ T     C     0     1         0         0         0
##  9 Pf3D~ 1110~ T     C     0     1         0         0         0
## 10 Pf3D~ 1120~ G     A     0     1         0         0         0
## # ... with 45 more rows, and 1 more variable: `PA0007-C` <chr>
```

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!

ggplot

Really, the package is `ggplot2` written by Hadley Wickham and is a series of functions that follow a specific data visaulization philosophy based on the "Grammar of Graphics".

Grammar of Graphics

The Grammar of Graphics by Leland Wilkinson

- Goal was to have a consistent "language" around graphics.
- This boils down to a "recipe", where every graph has a few key components:
  1. Data (to plot)
  2. A way to represent the points (geoms)
  3. Coordinate system

ggplot

In Hadley's own words. . .

Usage

It's hard to succinctly describe how ggplot2 works because it embodies a deep philosophy of visualisation. However, in most cases you start with `ggplot()`, supply a dataset and aesthetic mapping (with `aes()`). You then add on layers (like `geom_point()` or `geom_histogram()`), scales (like `scale_colour_brewer()`), faceting specifications (like `facet_wrap()`) and coordinate systems (like `coord_flip()`).

```
library(ggplot2)
```

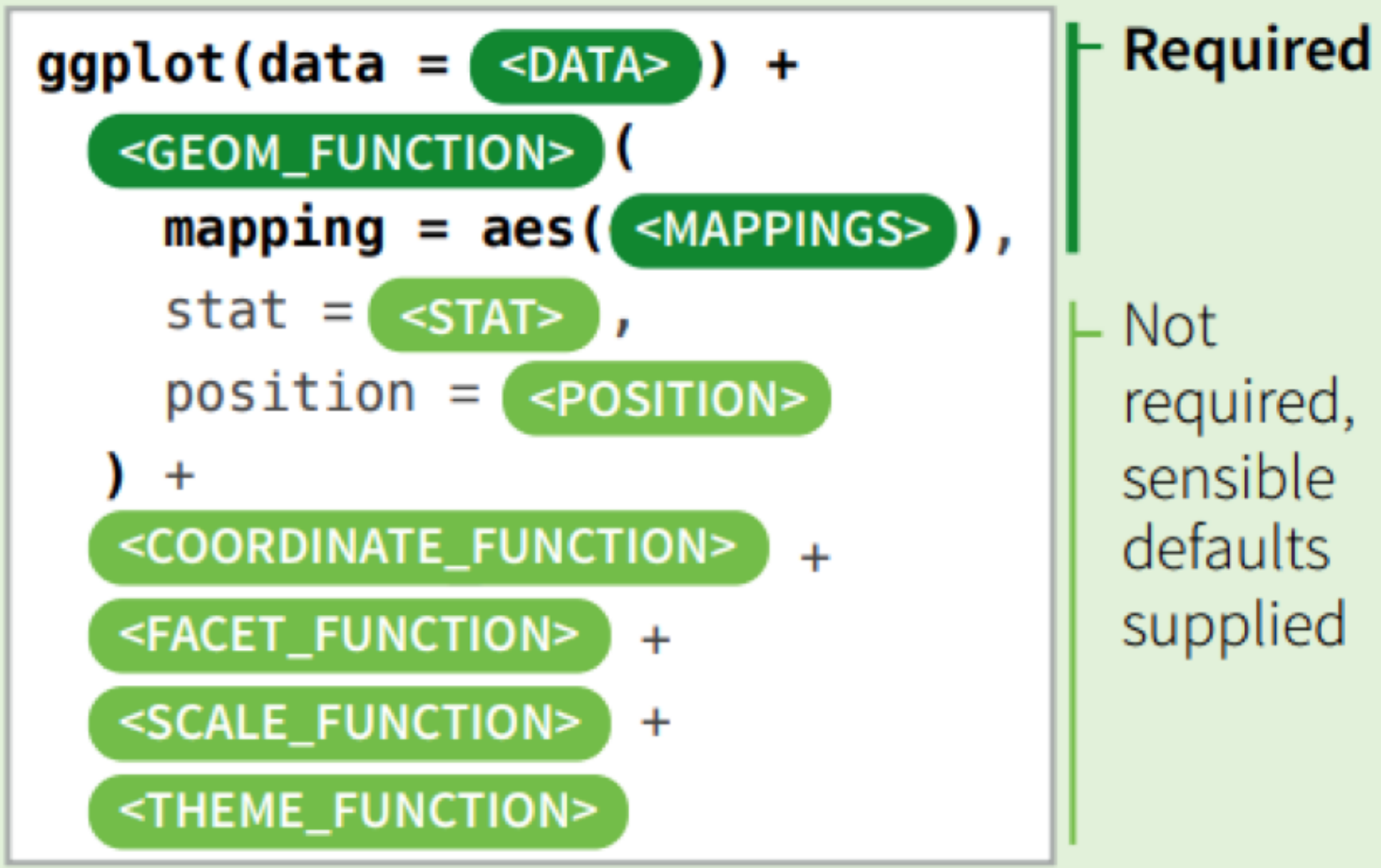
Let's Get Started (1)

You can think of a `ggplot2` layer as containing 5 main elements: 1. data [source] 2. mapping = aes() [what we want to plot] 3. geom [shape/characteristics of our plot] 4. stat [statistical transformations to do to the data] 5. position [adjust position characteristics of geom]*

- Personally, I rarely use position except for `jitter` (until I learned geom_jitter)

Let's Get Started (2)

Complete the template below to build a graph.



ggplot prefers data to be in the "long" format (see above).

Digging into Code (1)

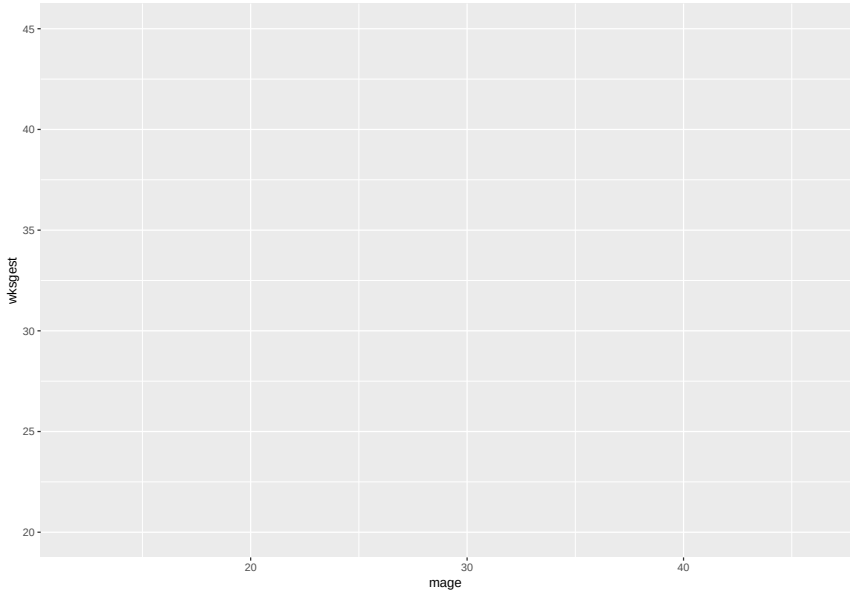
First we start with `ggplot` which creates a coordinate system for use to add our layer to (default cartesian coordinates – distance from origin, usually 0,0 or the min of your data).

```
ggplot2::ggplot(data=births)
```

Digging into Code (2)

Next, we need to pick what variables we actually want to map. From ggplot2, "Aesthetic mappings describe how variables in the data are mapped to visual properties (aesthetics) of geoms. Aesthetic mappings can be set in ggplot2() and in individual layers."

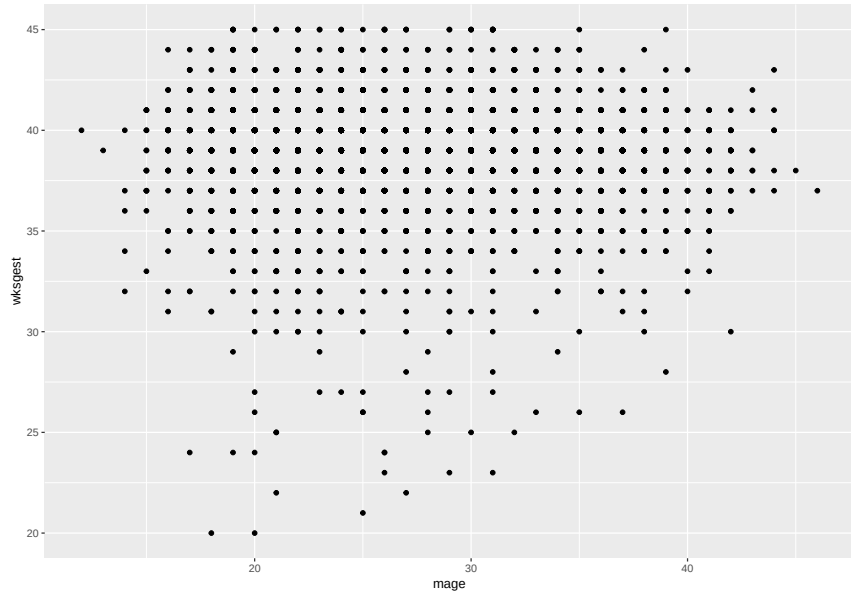
```
ggplot2::ggplot(data=births,
                aes(x=mage, y=wksgest))
```



Digging into Code (3)

Now, we need to pick how we want our variables to be plotted – i.e. what kind of graph.

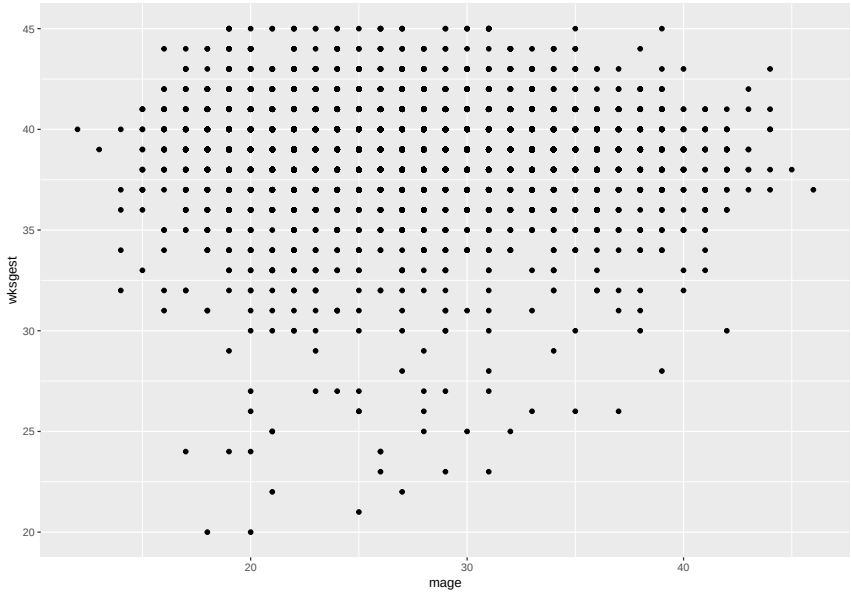
```
ggplot2::ggplot(data=births,
                aes(x=mage, y=wksgest)) +
  ggplot2::geom_point()
```



Digging into Code (4/5)

By default, `stat` is equal to "identity" for the geom_point geom. This means do not transform my data. Similarly, by default, the `position` is equal to "identity" for the geom_point geom. This means do not move my data (other options include jitter and dodge).

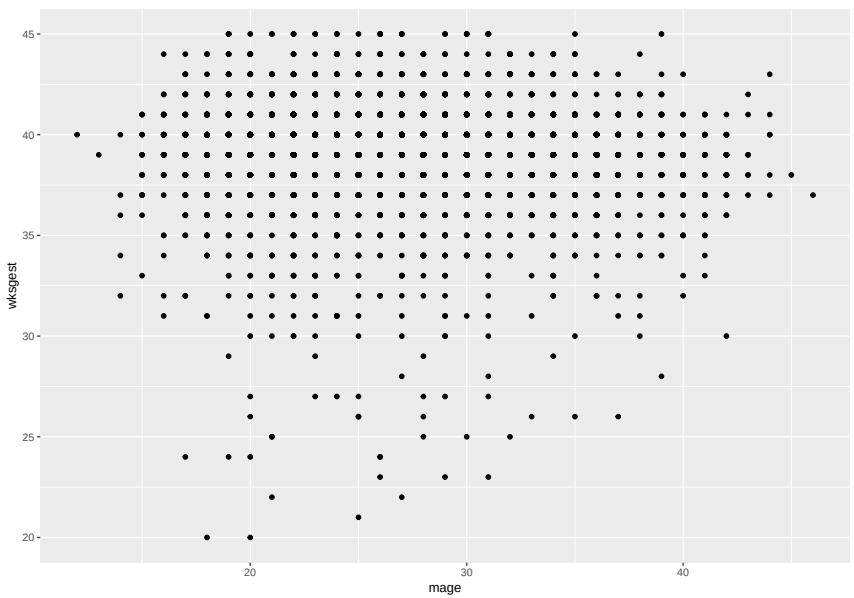
```
ggplot2::ggplot(data=births,
                aes(x=mage, y=wksgest)) +
  ggplot2::geom_point()
```



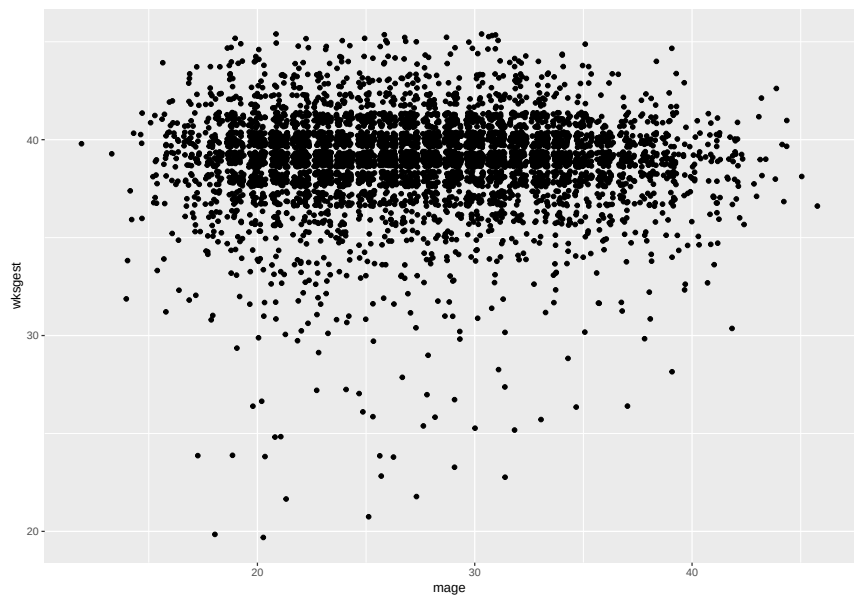
Digging into Code geoms

When you pick a `geom` someone has made the `position` and the `stat` decision for you (typically). Personally, the only time I have opened up the `layer` function and made my own layers if for package development (but would love to hear other examples!). For example, compare `geom_point` and `geom_jitter` which have different `position` arguments (by default).

```
ggplot2::ggplot(data=births,
                aes(x=mage, y=wksgest)) +
  ggplot2::geom_point()
```



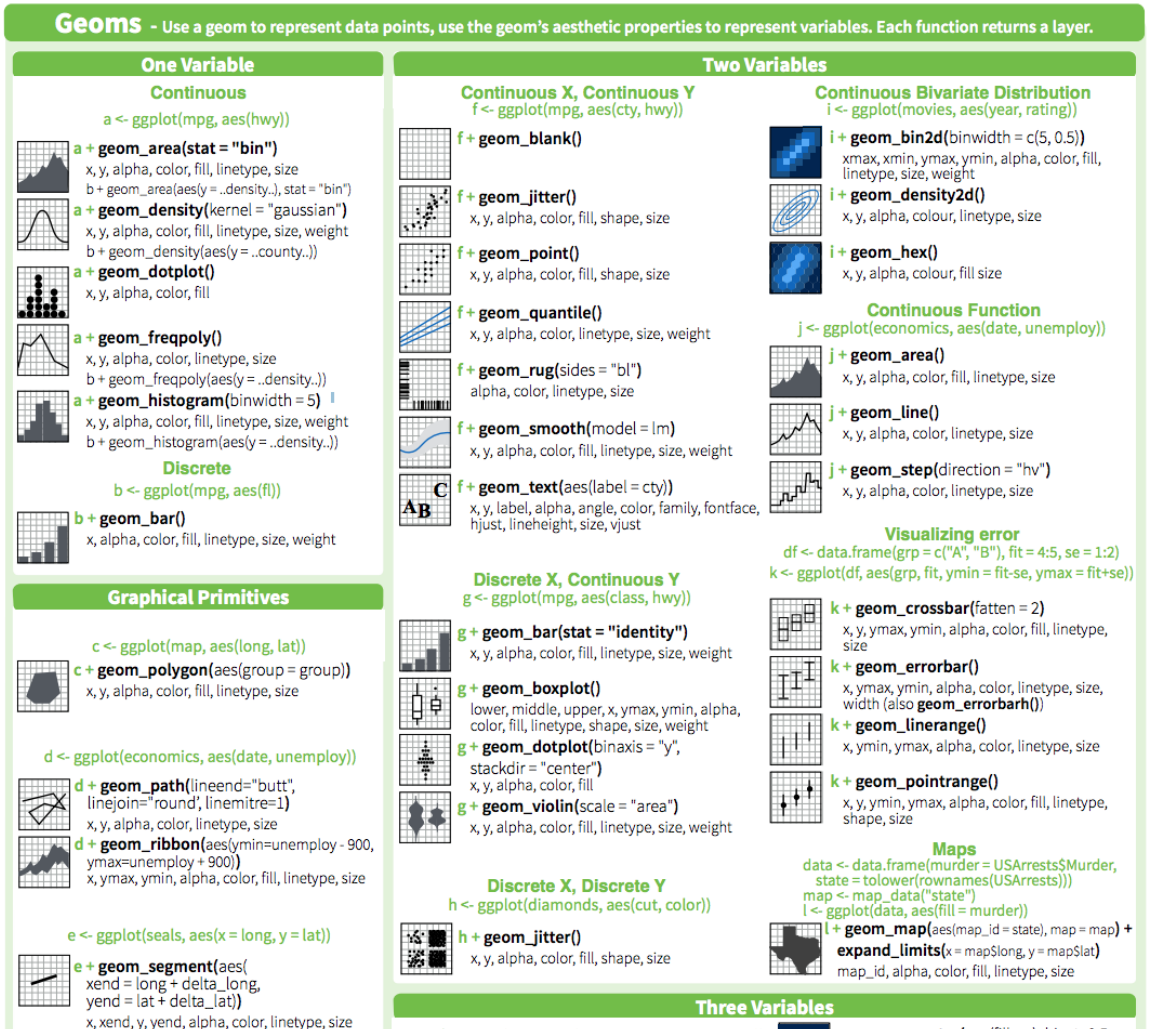

```
ggplot2::ggplot(data=births,
                aes(x=mage, y=wksgest)) +
  ggplot2::geom_jitter()
```



Geoms

There are more geoms than I can count and more coming out every day as R-developers push out new packages. We will play with some of these new geoms on Wednesday.

ggplot2 cheatshet



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

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

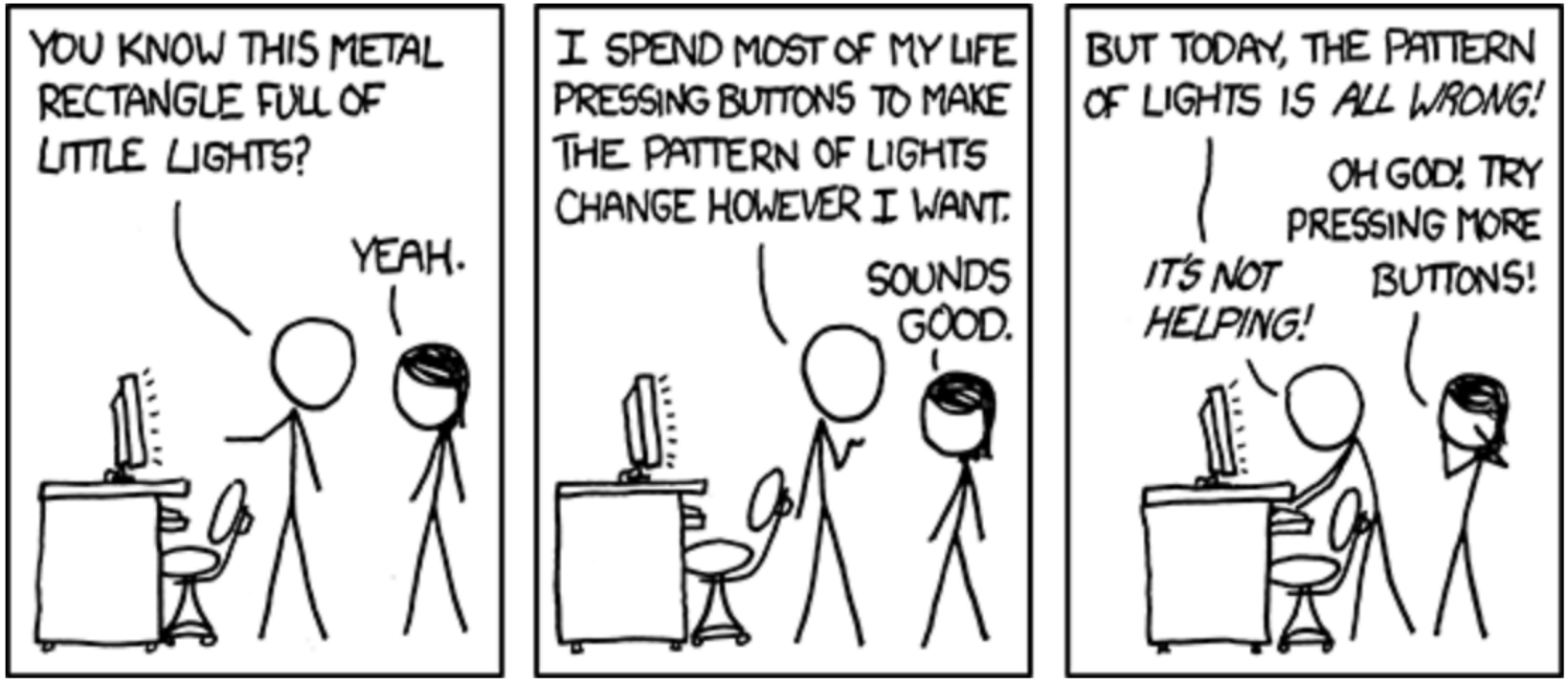
Three Variables

Geoms

These are the main geoms (thank you Garrick Adenbuie for this Figure).

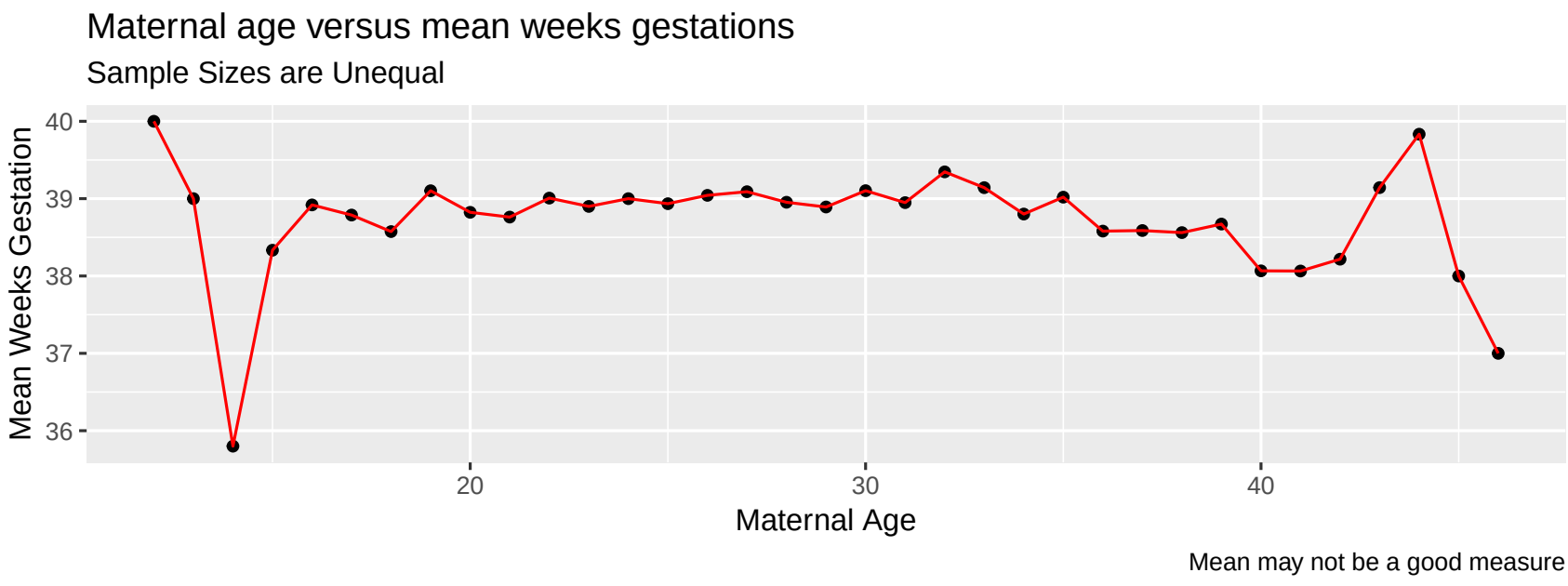
Type	Function
Point	`geom_point()`
Line	`geom_line()`
Bar	`geom_bar()`, `geom_col()`
Histogram	`geom_histogram()`
Regression	`geom_smooth()`
Boxplot	`geom_boxplot()`
Text	`geom_text()`
Vert./Horiz. Line	`geom_{vh}line()`
Count	`geom_count()`
Density	`geom_density()`

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!

ggplot objects are appendable

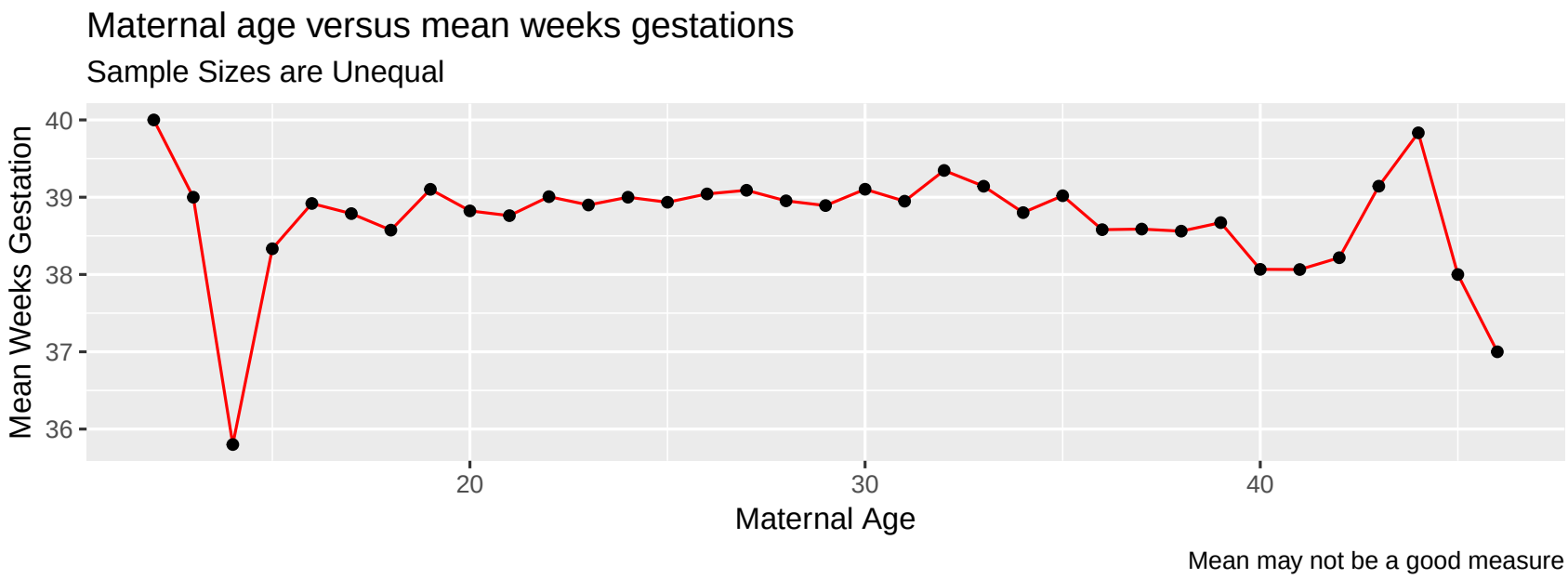
```
births %>%
  dplyr::group_by(mage) %>%
  dplyr::summarise(meanwksgest = mean(wksgest)) %>%
  ggplot2::ggplot(data=., aes(x=mage, y=meanwksgest)) +
  ggplot2::geom_point() +
  ggplot2::geom_line(color="red") +
  xlab("Maternal Age") +
  ylab("Mean Weeks Gestation") +
  ggtitle("Maternal age versus mean weeks gestations") +
  ggplot2::labs(subtitle = "Sample Sizes are Unequal",
                caption = "Mean may not be a good measure")
```



ggplot objects are appendable

Note the difference here in which layer is called first!

```
births %>%
  dplyr::group_by(mage) %>%
  dplyr::summarise(meanwksgest = mean(wksgest)) %>%
  ggplot2::ggplot(data=., aes(x=mage, y=meanwksgest)) +
  ggplot2::geom_line(color="red") +
  ggplot2::geom_point() +
  xlab("Maternal Age") +
  ylab("Mean Weeks Gestation") +
  ggtitle("Maternal age versus mean weeks gestations") +
  ggplot2::labs(subtitle = "Sample Sizes are Unequal",
                caption = "Mean may not be a good measure")
```



Preferred practice

If we use the `ggplot` function, we automatically pass the (. . .) arguments to each geom. However, if I know I have mutliple datasets that I would like to plot layer, then I will use the `data` call for each geom.

```
magewksgest <- births %>%
  dplyr::group_by(mage) %>%
  dplyr::summarise(n=n(), meanwksgest = mean(wksgest))

ggplot2::ggplot() +
  ggplot2::geom_point(data=births, aes(x=mage, y=wksgest)) +
  ggplot2::geom_line(data=magewksgest, aes(x=mage, y=meanwksgest), color = "red")
```

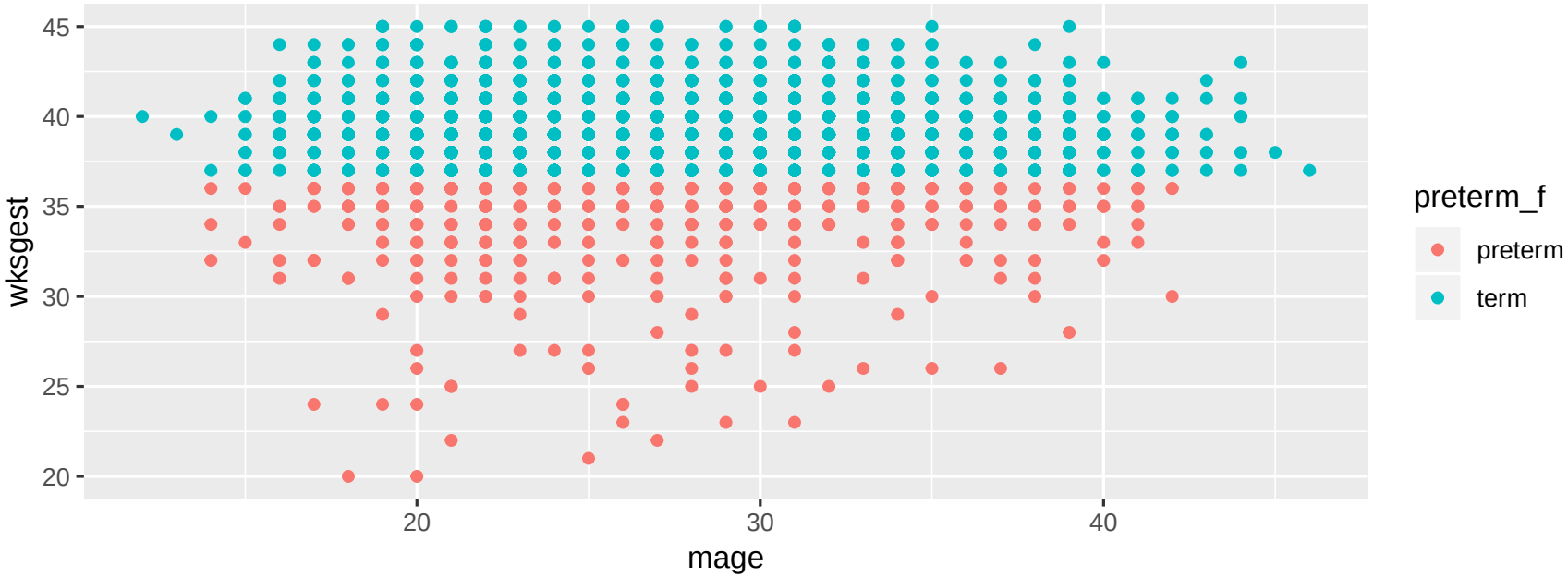
Digging into Aesthetics (1)

More R for Data Scientist (there is a theme here), "An aesthetic is a visual property of the objects in your plot. Aesthetics include things like the size, the shape, or the color of your points. You can display a point (like the one below) in different ways by changing the values of its aesthetic properties."

So far we have only really thought about `aesthetics` as the X and Y variables we would like to plot (on our cartesian graph). What if we wanted them to have additional features by a third variable?

Digging into Aesthetics (2)

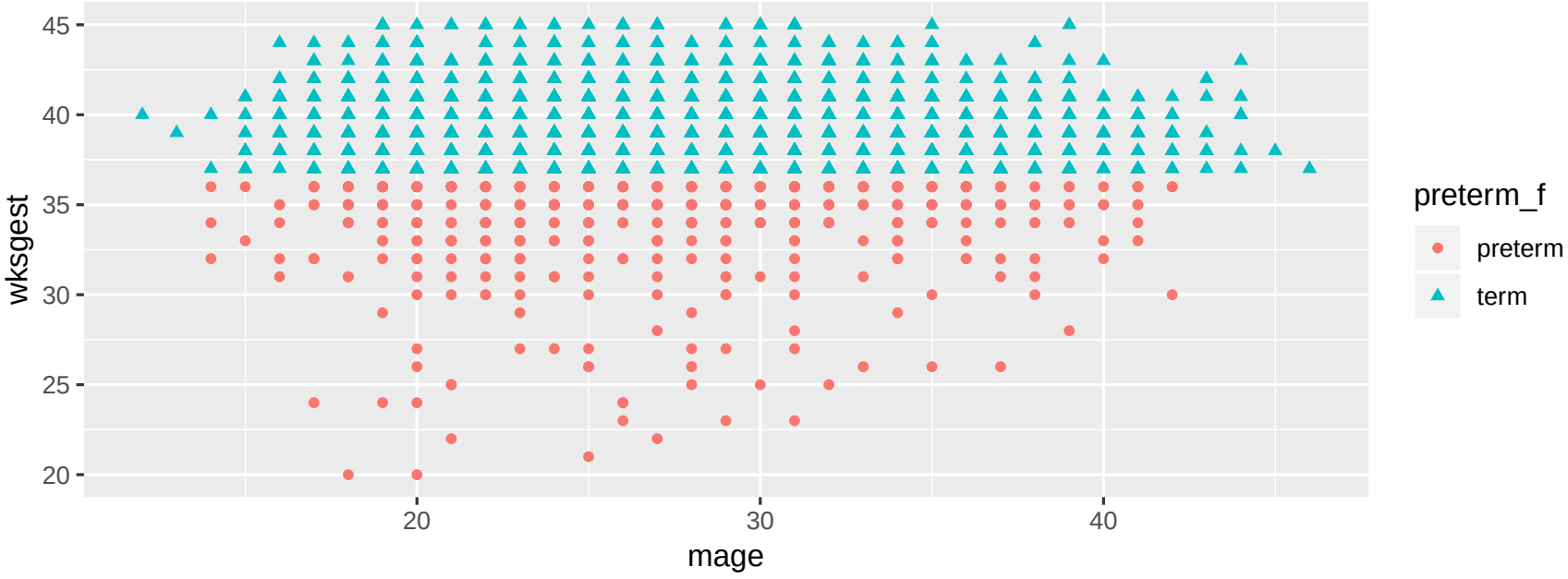
```
births %>%
  ggplot2::ggplot() +
  ggplot2::geom_point(data=births, aes(x=mage, y=wksgest, color=preterm_f))
```




```
# note use of factor here. R asssume factors are disjoint indicators/categorical vars
# R assumes characters and numerics are (maybe) continous by default
```

Digging into Aesthetics (3)

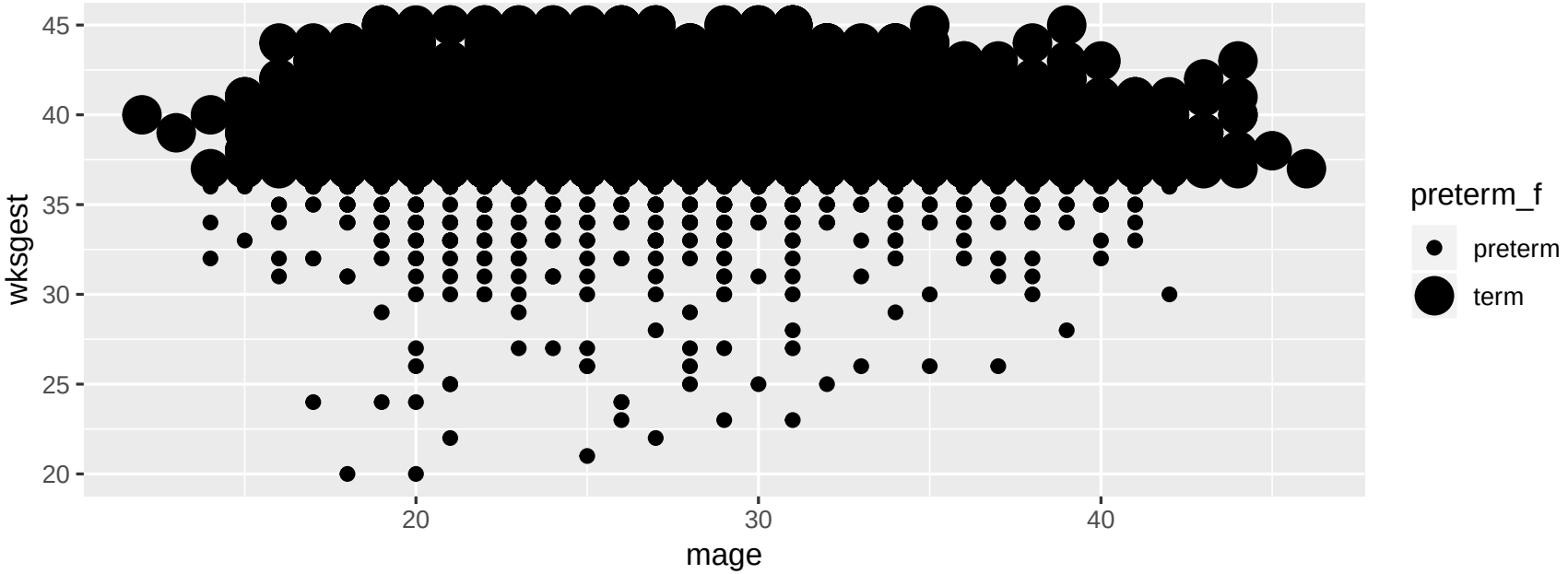
```
births %>%
  ggplot2::ggplot() +
  ggplot2::geom_point(data=births, aes(x=mage, y=wksgest,
                                       colour=preterm_f, shape=preterm_f))
```




```
# note use of factor here. R asssume factors are disjoint indicators/categorical vars
# R assumes characters and numerics are (maybe) continous by default
```

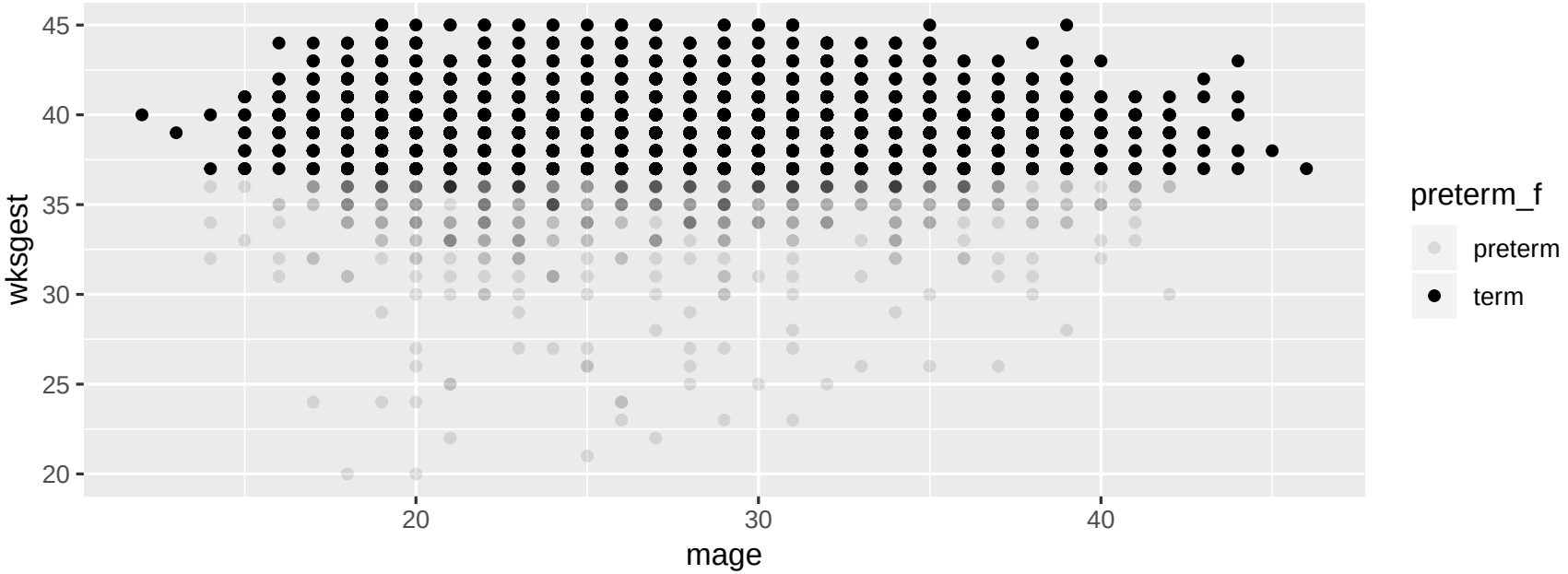
Digging into Aesthetics (4)

```
ggplot2::ggplot() +
  ggplot2::geom_point(data=births,
                      aes(x=mage, y=wksgest, size=preterm_f))
```



Digging into Aesthetics (5)

```
ggplot2::ggplot() +
  ggplot2::geom_point(data=births,
                      aes(x=mage, y=wksgest, alpha=preterm_f))
```



Differentiating Aes from geom properties (1)

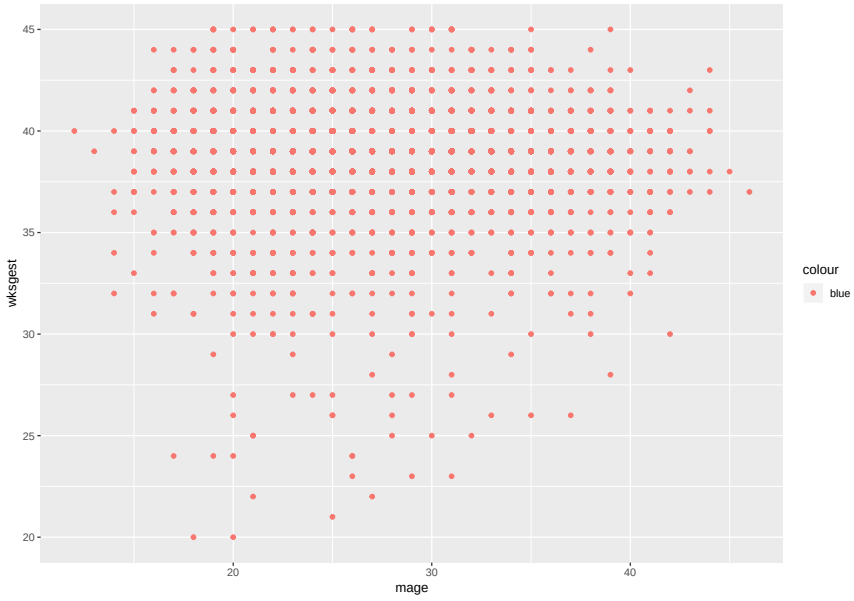
You may have noticed that I have used color inside and outside of the `aes()`. This is an **important** distinction. When different properties (i.e. color, shape) are put inside the `aes(colour = <var>)` and assigned a variable, R interprets this as "You want me to change the color (shape, etc) dependent on this covariate".

This is contrasted by the code `<geom>(aes(...) + colour = "red")` where we have assigned a specific propoerty to the entire geom object.

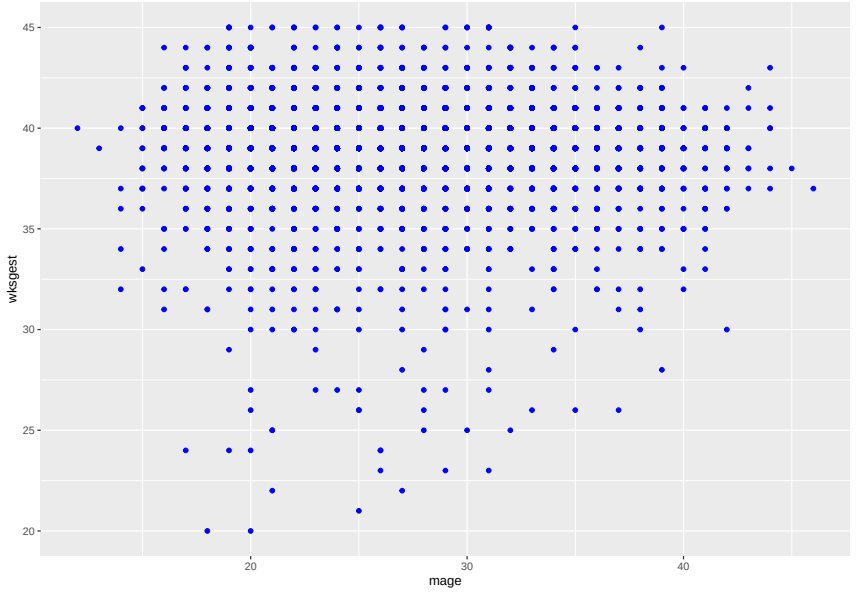
Let's take a look.

Differentiating Aes from geom properties (2)

```
ggplot2::ggplot(data=births,
                aes(x=mage, y=wksgest,
                    color = "blue")) +
  ggplot2::geom_point()
```



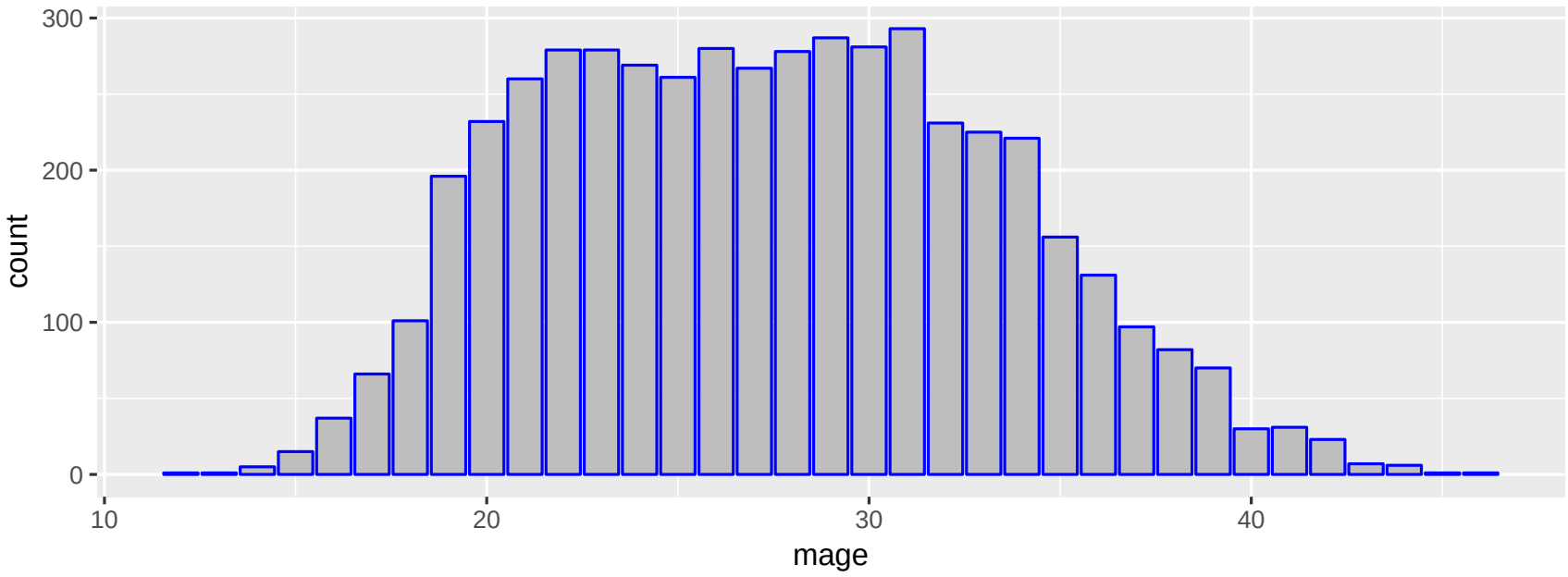

```
ggplot2::ggplot(data=births,
                aes(x=mage, y=wksgest)) +
  ggplot2::geom_point(color="blue")
```



Differentiating Aes from geom properties (3)

It is also worth noting that there is a difference between the `fill` and `color` options for geoms. This is important for many geoms.

```
ggplot2::ggplot(data=births,
                aes(x=mage)) +
  ggplot2::geom_bar(color="blue", fill="grey")
```

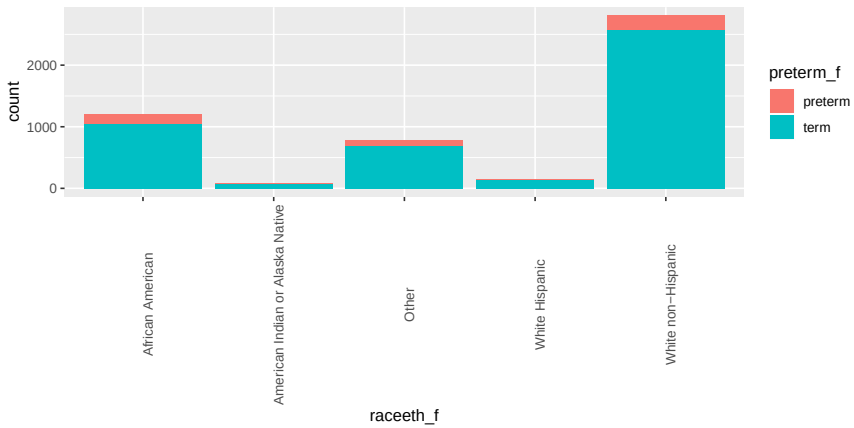


Digging into Positions (1)

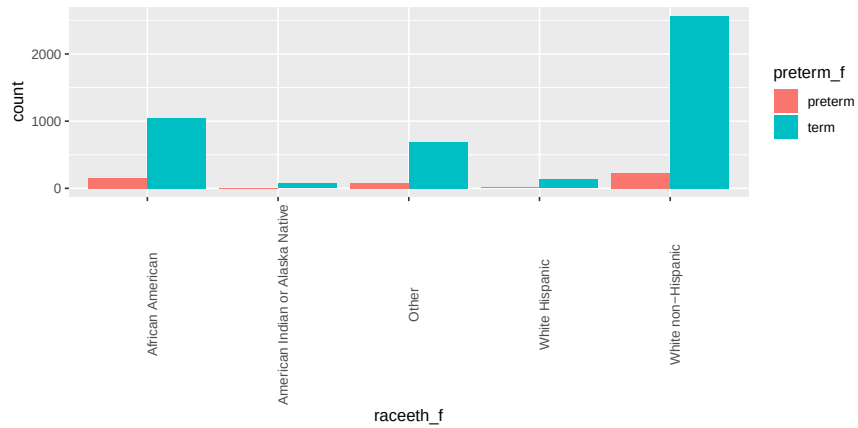
The "position" argument is specific for many geoms. However, bar graphs (and a few other geoms) have a couple of "nuances" (see rich discussion R for Data Scienc). We will explore these briefly.

Digging into Positions (2)

```
ggplot2::ggplot(data=births,
                aes(x=raceeth_f,
                    fill=preterm_f)) +
  ggplot2::geom_bar() +
  theme(axis.text.x = element_text(angle=90))
```

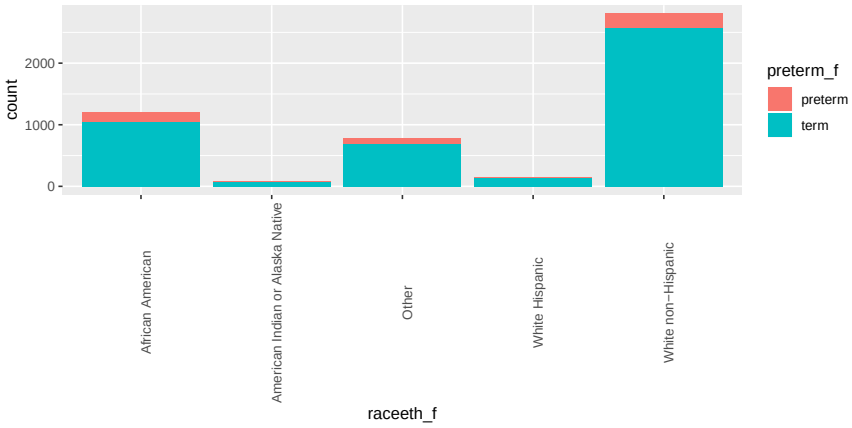



```
ggplot2::ggplot(data=births,
                aes(x=raceeth_f,
                    fill=preterm_f)) +
  ggplot2::geom_bar(position="dodge") +
  theme(axis.text.x = element_text(angle=90))
```

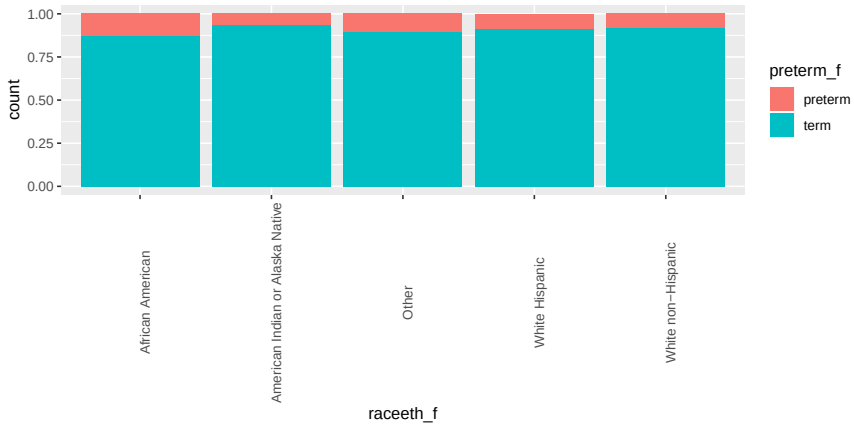


Digging into Positions (3)

```
ggplot2::ggplot(data=births,
                aes(x=raceeth_f,
                    fill=preterm_f)) +
  ggplot2::geom_bar() +
  theme(axis.text.x = element_text(angle=90))
```




```
ggplot2::ggplot(data=births,
                aes(x=raceeth_f,
                    fill=preterm_f)) +
  ggplot2::geom_bar(position="fill") +
  theme(axis.text.x = element_text(angle=90))
```



Digging into Statistical Transformations (1)

This is a bit more of a nuanced topic, as geoms typically have predifined statistical transformations (many times "identity") that we can override. The `stat="identity"` code block works because we of the way R interprets layers (see `?layer`) but it is really a function: `stat_identity`. So, if you want to know what the stat is doing (i.e. help documentation), you will need to call the `?<function>`

In additoin, we can actually map stats to our regular aesthetic using ".." notation. For example, this is commonly done when you want to change a count to a proportion (..prop..). From R for Data Scientist, "The two dots that surround prop notify ggplot2 that the prop variable appears in the transformed data set that is created by the stat, and not the raw data set. Be sure to include these dots whenever you refer to a variable that is created by a stat.""

Digging into Statistical Transformations (2)

There are 22 stats for use . . . R for Data Scientist lists them here:

Stats

A stat transforms raw data into new variables to plot.
Each stat is listed with the parameters it takes
and the variables it creates (surrounded by ..).

stat_bin() - Bins and counts continuous data.
bins, binwidth, breaks, drop, origin, right, width
..count.., ..ncount.., ..density.., ..ndensity..

stat_bin_2d() - Bins and counts 2D continuous data.
bins, binwidth, drop
..count.., ..density..

stat_bin_hex() - 2D bins with hexagonal shape.
bins, binwidth
..count.., ..density..

stat_boxplot() - Computes boxplot parameters.
coef
..lower.., ..middle.., ..notchlower.., ..notchupper..,
..upper.., ..width.., ..ymin.., ..ymax..

stat_contour() - Computes contours of 3D surface.
..level..

stat_count() - Counts discrete data.
width
..count.., ..prop..

stat_density() - Computes density kernel estimate.
adjust, kernel, trim
..count.., ..density.., ..scaled..

stat_density_2d() - 2D density kernel estimate.
contour, n, h
..level..

stat_ecdf(n = 40) - Computes empirical CDF.
..x.., ..y..

stat_ellipse() - Computes model ellipses.
level, segments, t

stat_function() - Applies a function to x variable.
args, fun, n
..x.., ..y..

stat_identity() - Returns data as is

stat_qq() - Calculates quantile quantile plot.
distribution, dparams
..sample.., ..theoretical..

stat_quantile() - Computes quantiles.
formula, quantiles, method, method.args
..quantile..

stat_smooth() - Computes model line.
formula, fullrange, level, method, n, se, span
..se.., ..x.., ..y.., ..ymin.., ..ymax..

stat_sum() - Counts discrete data.
..n.., ..prop..

stat_summary() - Applies summary to groups of unique x values.
fun.args, fun.data, fun.y, fun.ymin, fun.ymax

stat_summary_2d() - Applies summary to groups of 2D binned values.
bins, binwidth, drop, fun, fun.args
..value..

stat_summary_bin() - Applies summary to groups of binned x values.
fun.args, fun.data, fun.y, fun.ymin, fun.ymax

stat_summary_hex() - Applies summary to groups of 2D hexagonally binned values.
bins, binwidth, drop, fun, fun.args
..value..

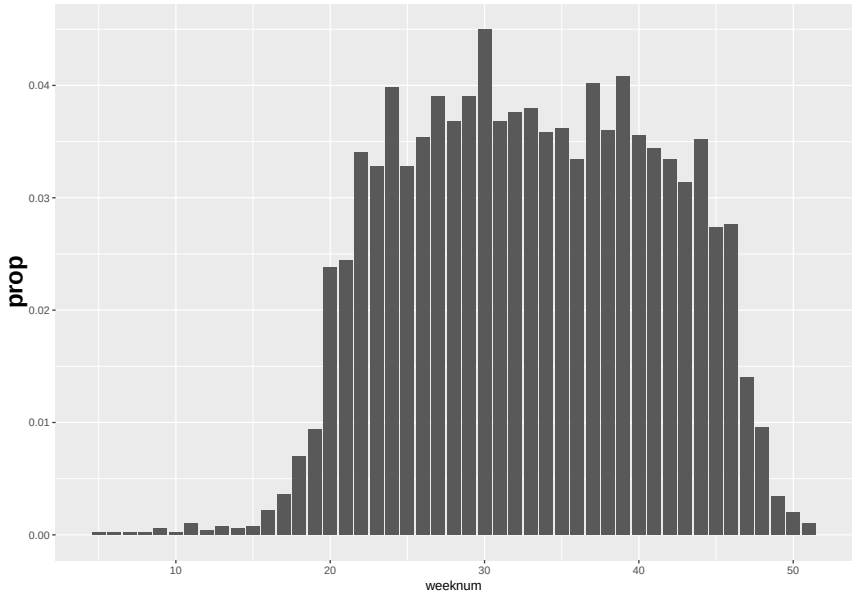
stat_unique() - Removes duplicates.

stat_ydensity() - Computes densities for violin plot.
..count.., ..density.., ..n.., ..scaled.., ..violinwidth.., ..width..

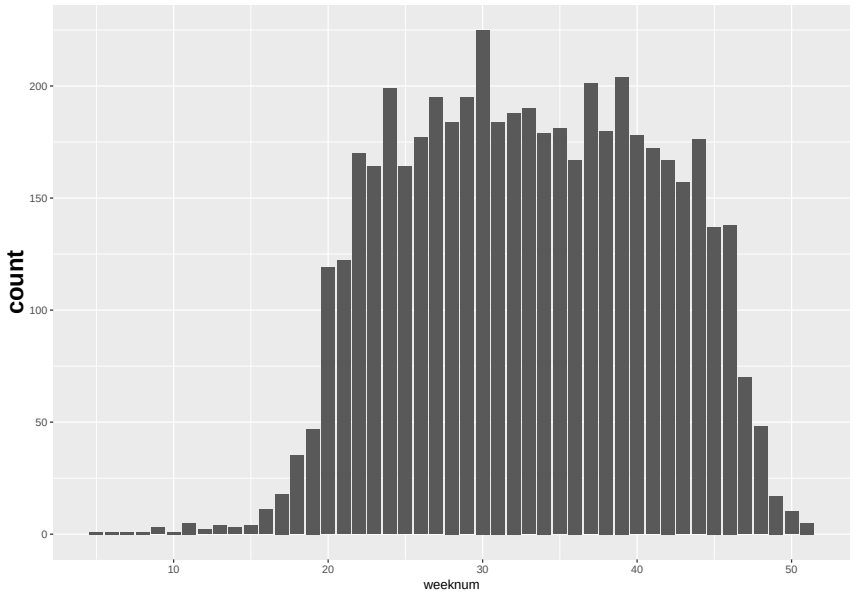
Digging into Statistical Transformations (3)

Homework 3, Question (hard part)

```
ggplot(births, aes(x=weeknum, y=..prop..)) +
  geom_bar() +
  theme(axis.title.y =
          element_text(size=20, face="bold"))
```




```
ggplot(births, aes(x=weeknum)) +
  geom_bar() +
  theme(axis.title.y =
          element_text(size=20, face="bold"))
```



Additonal Important Features of ggplot

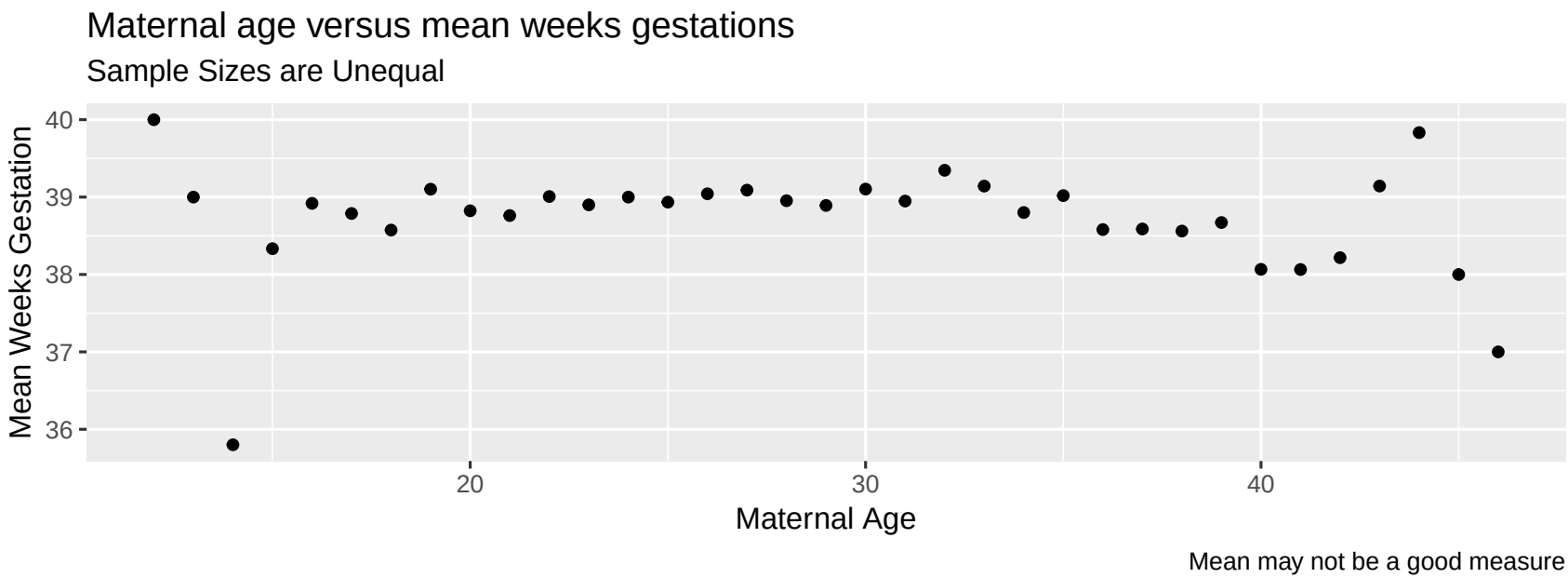
- *Labels* to control the title and axis labels.

Wednesday

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

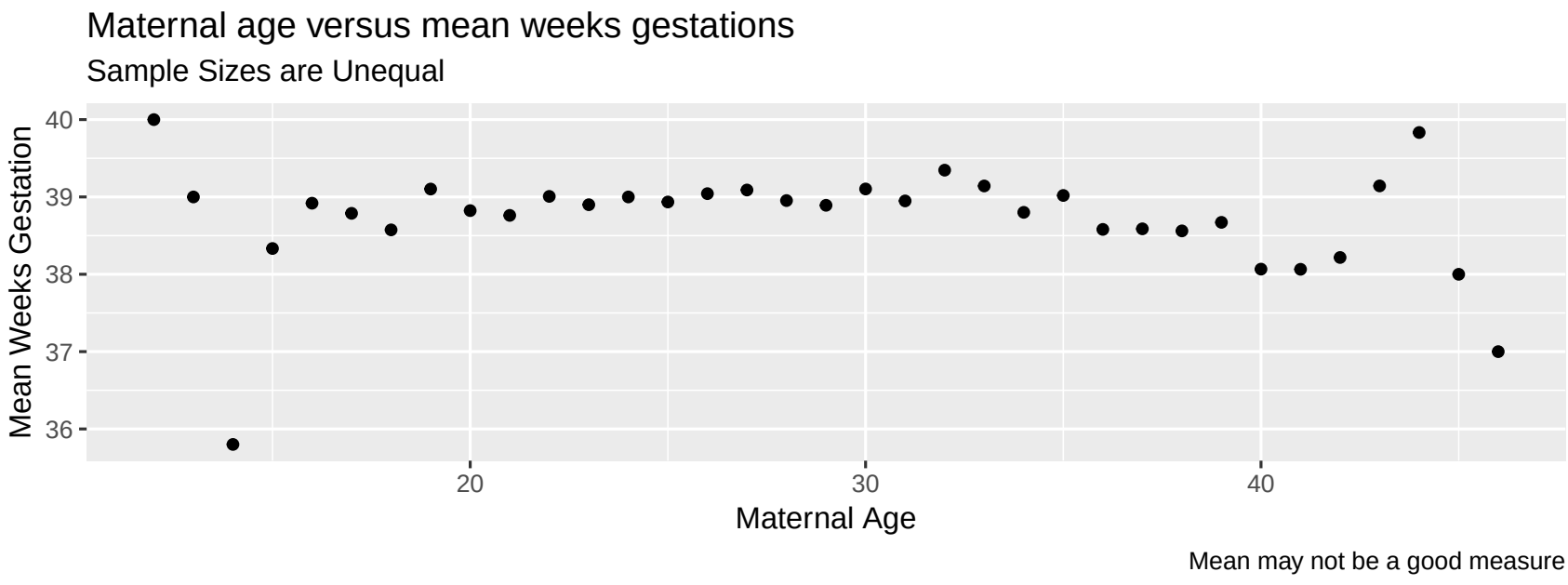
Labels

```r
# note differences between %>% and +
births %>%
  dplyr::group_by(mage) %>%
  dplyr::summarise(meanwksgest = mean(wksgest)) %>%
  ggplot2::ggplot(data=., aes(x=mage, y=meanwksgest)) +
  ggplot2::geom_point() +
  ggplot2::labs(title="Maternal age versus mean weeks gestations",
                subtitle = "Sample Sizes are Unequal",
                caption = "Mean may not be a good measure",
                x = "Maternal Age",
                y = "Mean Weeks Gestation")
```



Labels (Equivalent approach)

```r
# note differences between %>% and +
births %>%
  dplyr::group_by(mage) %>%
  dplyr::summarise(meanwksgest = mean(wksgest)) %>%
  ggplot2::ggplot(data=., aes(x=mage, y=meanwksgest)) +
  ggplot2::geom_point() +
  xlab("Maternal Age") +
  ylab("Mean Weeks Gestation") +
  ggtitle("Maternal age versus mean weeks gestations") +
  ggplot2::labs(subtitle = "Sample Sizes are Unequal",
                caption = "Mean may not be a good measure")
```



Additonal Important Features of ggplot

- *Labels* to control the title and axis labels.

For Wednesday

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

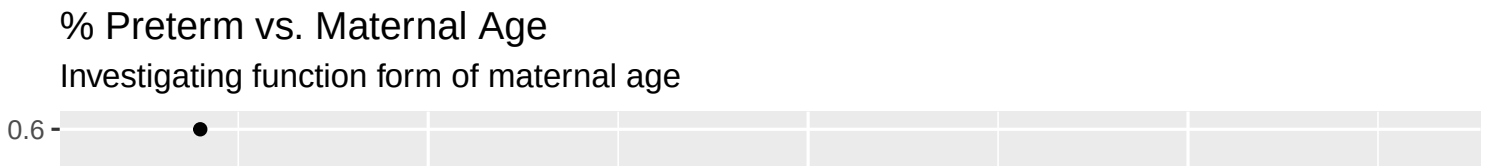
Pulling it all together!

Homework 3, Question 2

```
mage_df = births %>% group_by(mage) %>% #(HW3.2.2) setup
  summarize(n=n(),
            pct_earlyPNC = mean(pnc5, na.rm=T),
            pct_preterm = mean(preterm, na.rm=T))
head(mage_df) #(HW3.2.2) answer
```

```
## # A tibble: 6 x 4
##    mage     n pct_earlyPNC pct_preterm
##   <int> <int>        <dbl>       <dbl>
## 1    12     1        1           0
## 2    13     1        1           0
## 3    14     5        0.6         0.6
## 4    15    15        0.538       0.133
## 5    16    37        0.838       0.135
## 6    17    66        0.828       0.136
```

```
#(HW3.2.3A)
ggplot(mage_df, aes(mage, 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.")
```



Pulling it all together!

Check out geom_smooth

It is an incredible function that allows for a lot of flexibility!!

```
# ?geom_smooth
```

Conclusion

The graphs we made today were kind-of pretty? Wednesday we are going to be focusing a lot on making customized, gorgeous figures with scales, facetting, themes, grids, etc.
We will also be looking at how to make figures interactive (ridiculously easy thanks to `plotly`).