Introduction to ggplot2

Victor Yuan

2020-07-09

Set up

Install these packages

```
install.packages(tidyverse)
```

Load libraries

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Load gene expression / methylation data

```
geo_data <- read_csv('https://raw.githubusercontent.com/wvictor14/TOG/master/data/GSE98224.csv')

geo_data
```

```
## # A tibble: 48 x 159
##    expr_geo_id meth_geo_id diagnosis tissue maternal_age maternal_bmi
##    <chr>       <chr>       <chr>     <chr>         <dbl>        <dbl>
##  1 GSM1940495  GSM2589532  PE        Place~           37         19.5
##  2 GSM1940496  GSM2589533  PE        Place~           40         25.7
##  3 GSM1940499  GSM2589534  PE        Place~           37         25
##  4 GSM1940500  GSM2589535  PE        Place~           38         26.2
##  5 GSM1940501  GSM2589536  PE        Place~           33         31.2
##  6 GSM1940502  GSM2589537  PE        Place~           26         31.2
##  7 GSM1940505  GSM2589538  PE        Place~           31         18.6
##  8 GSM1940506  GSM2589539  PE        Place~           37         25.2
##  9 GSM1940507  GSM2589540  non-PE    Place~           35         18.6
## 10 GSM1940508  GSM2589541  PE        Place~           32         26.6
## # ... with 38 more rows, and 153 more variables: maternal_ethnicity <chr>,
## #   ga_weeks <dbl>, ga_days <dbl>, transcript_8033795 <dbl>,
## #   transcript_8103881 <dbl>, transcript_7904014 <dbl>, transcript_8127692 <dbl>,
## #   transcript_7990031 <dbl>, transcript_8121144 <dbl>, transcript_8150846 <dbl>,
## #   transcript_7962246 <dbl>, transcript_7941890 <dbl>, transcript_7896644 <dbl>,
## #   transcript_7992897 <dbl>, transcript_7973002 <dbl>, transcript_7979800 <dbl>,
## #   transcript_8112007 <dbl>, transcript_8036686 <dbl>, transcript_8001325 <dbl>,
## #   transcript_8180328 <dbl>, transcript_8109283 <dbl>, transcript_8041223 <dbl>,
## #   transcript_8144703 <dbl>, transcript_7997556 <dbl>, transcript_7955896 <dbl>,
## #   transcript_7939897 <dbl>, transcript_8035078 <dbl>, transcript_8113094 <dbl>,
## #   transcript_7893397 <dbl>, transcript_8110708 <dbl>, transcript_8102610 <dbl>,
```

3 essential components

to every ggplot2 graph

Data, **Geom**etry, **Aes**thetics

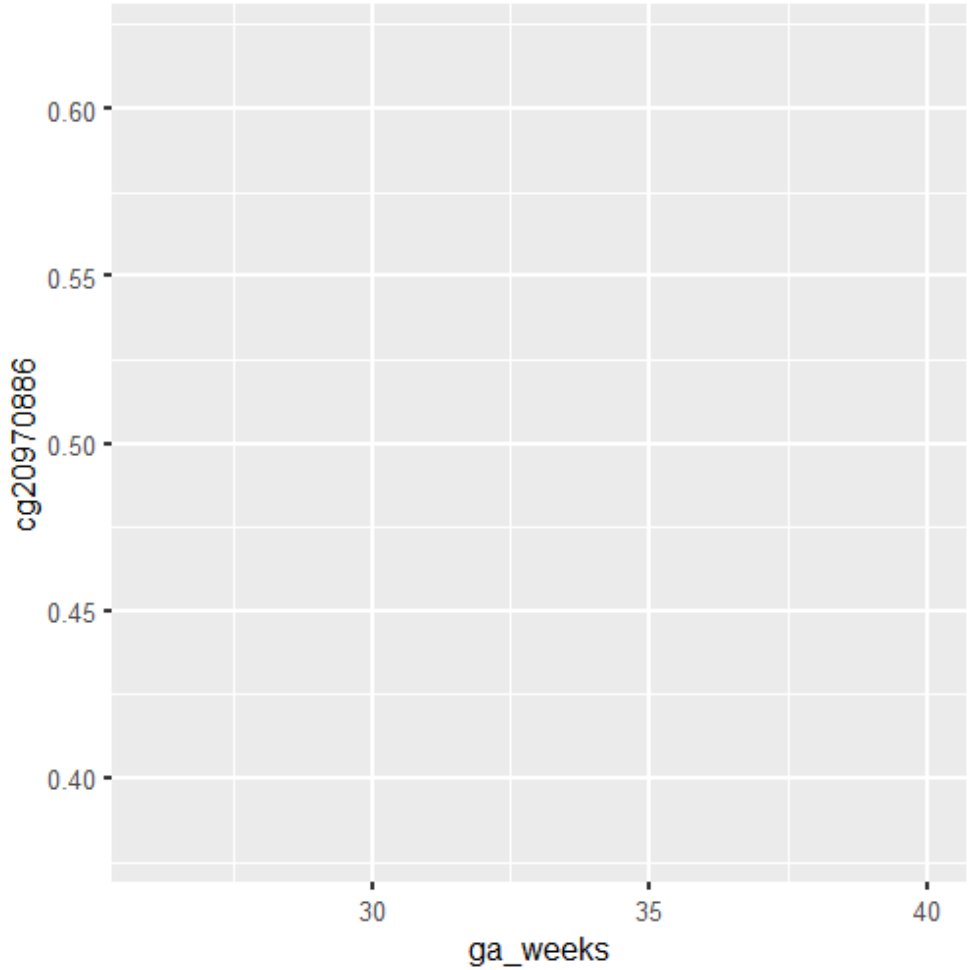
First step of every ggplot2 call is to *declare* the data.

```
ggplot(data = geo_data)
```

Then, we can assign variables in our data to different *aesthetics* of the plot.

```
ggplot(data = geo_data,
       aes(x = ga_weeks,
           y = cg20970886))
```

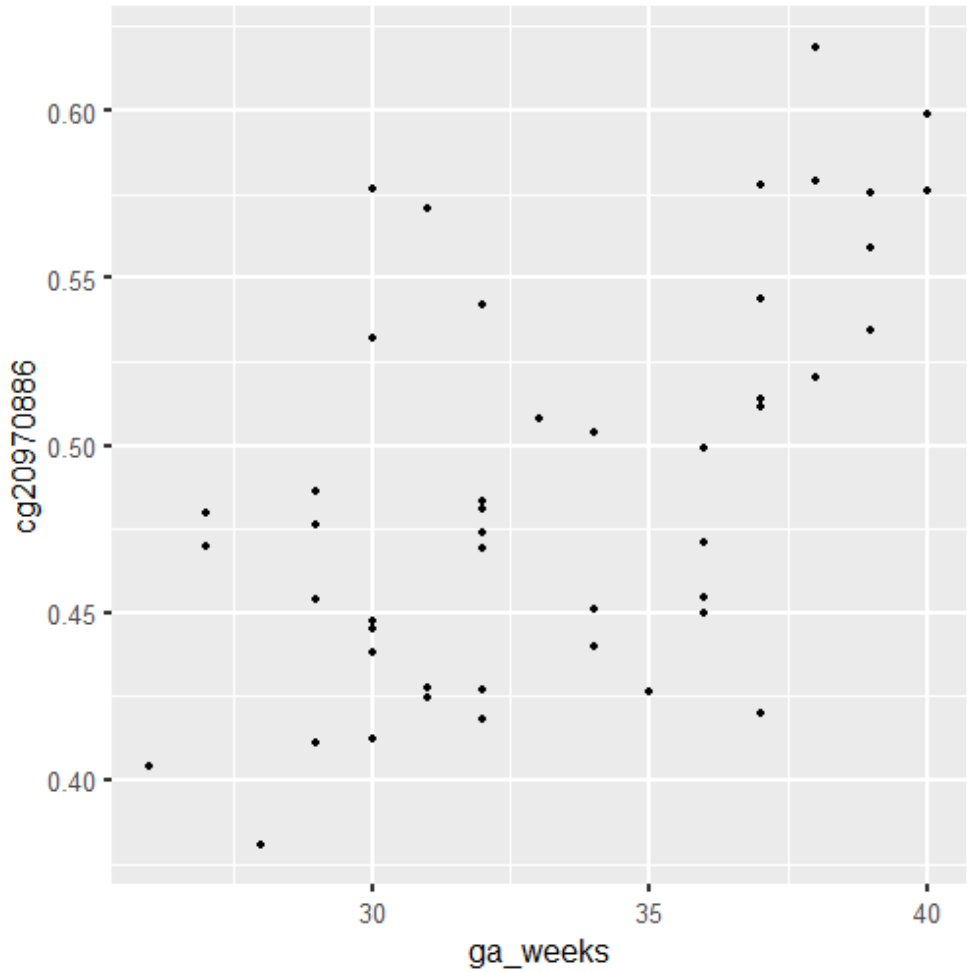
This is referred to as *aesthetic mapping*.



Add **geometries (geoms)** to complete the plot.

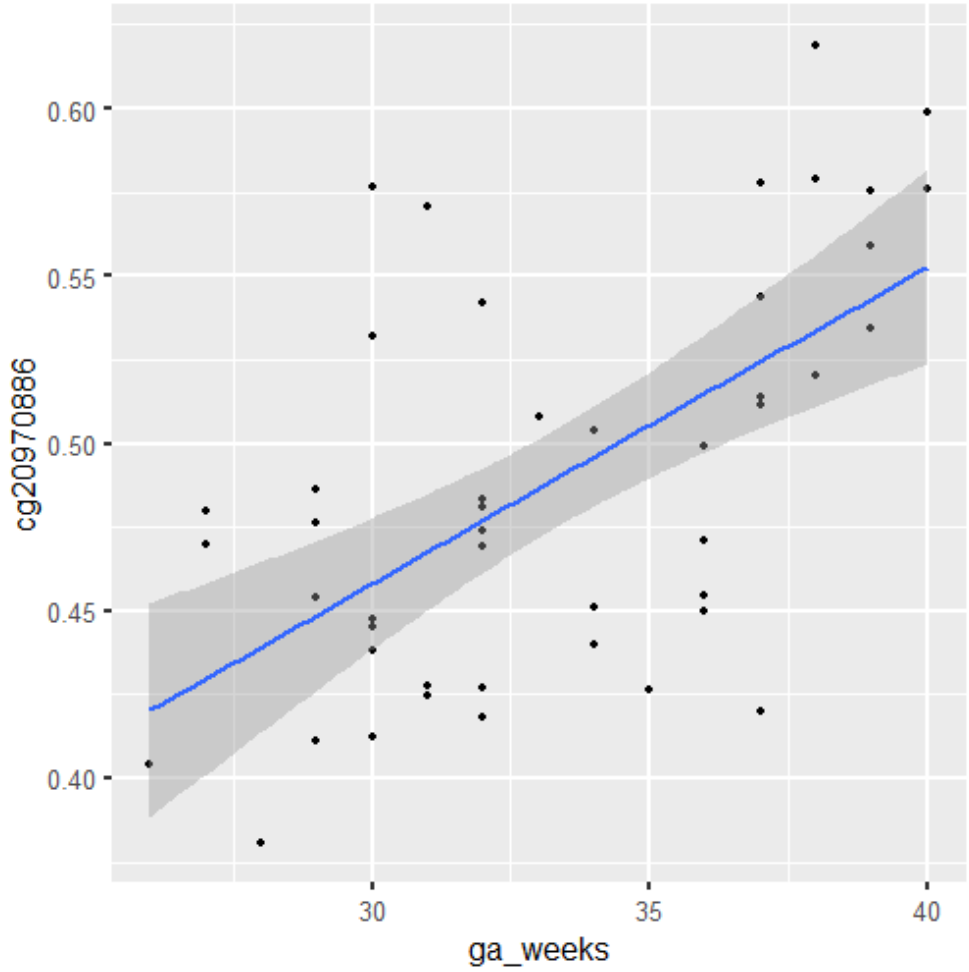
```
ggplot(data = geo_data,
       aes(x = ga_weeks,
           y = cg20970886)) +
  geom_point()
```

Geoms are like saying what type of plot you want (e.g. scatterplot, boxplots, histograms... etc.)



There are many *geoms*. Sometimes it makes sense to combine several.

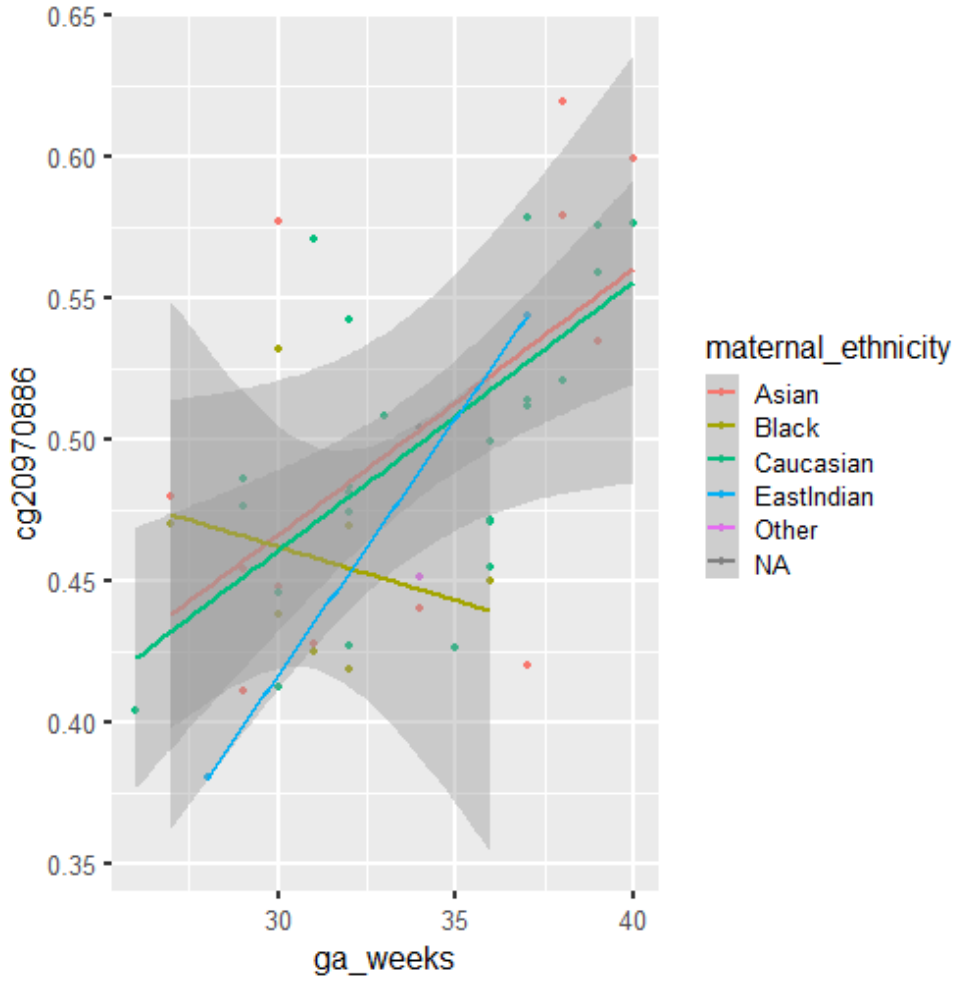
```
ggplot(data = geo_data,
       aes(x = ga_weeks,
           y = cg20970886)) +
  geom_point() +
  geom_smooth(method = "lm")
```



We can assign other variables to other aesthetics, e.g. color.

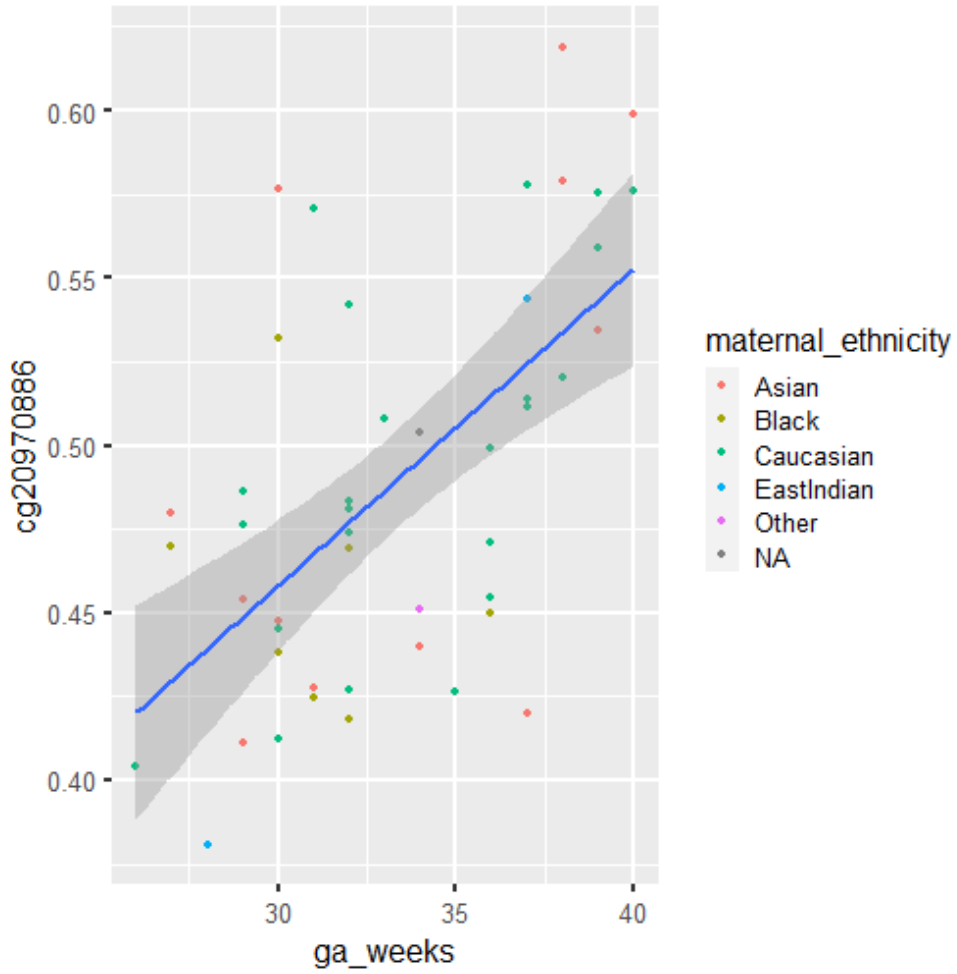
```
ggplot(data = geo_data,
       aes(x = ga_weeks,
           y = cg20970886,
           color = maternal_ethnicity)) +
  geom_point() +
  geom_smooth(method = "lm")
```

But note that this assigned maternal ethnicity to the color of both points and lines!



To assign color exclusively to points (and not lines), put inside specific geom:

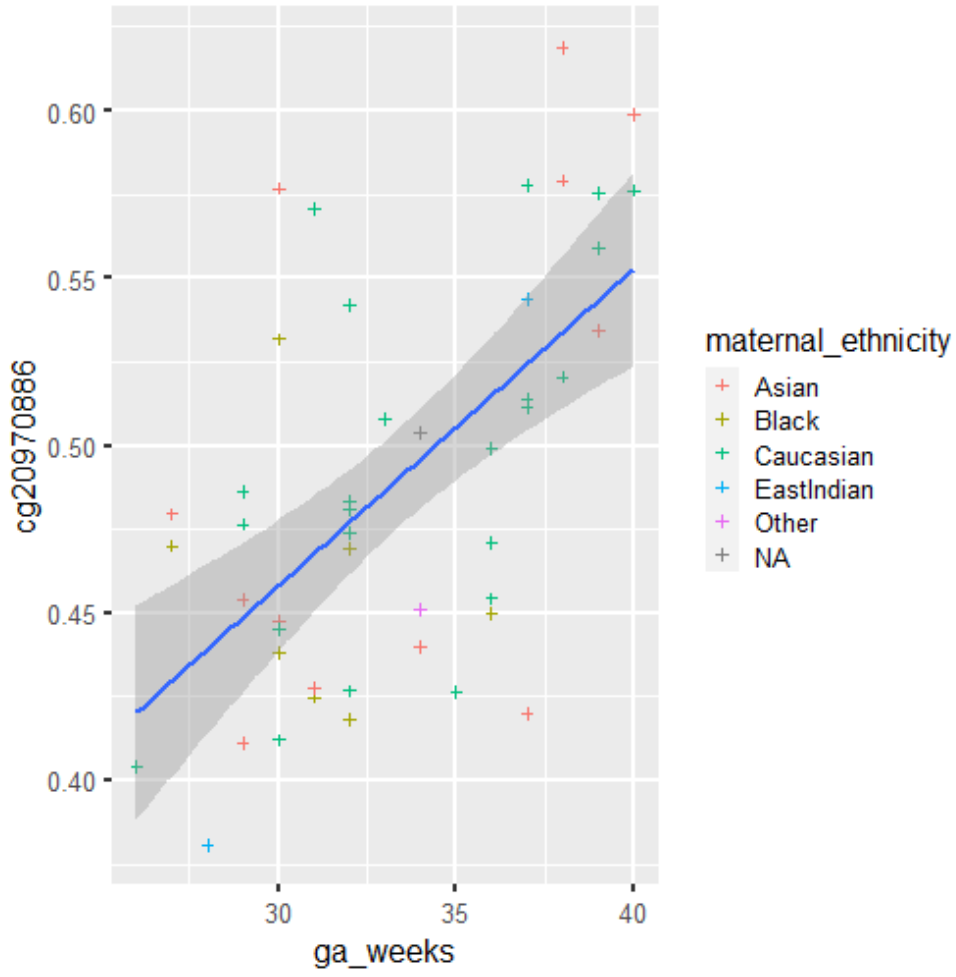
```
ggplot(data = geo_data,
       aes(x = ga_weeks,
           y = cg20970886)) +
  geom_point(aes(color = maternal_ethnicity)) +
  geom_smooth(method = "lm")
```



Can change the *shape* of points

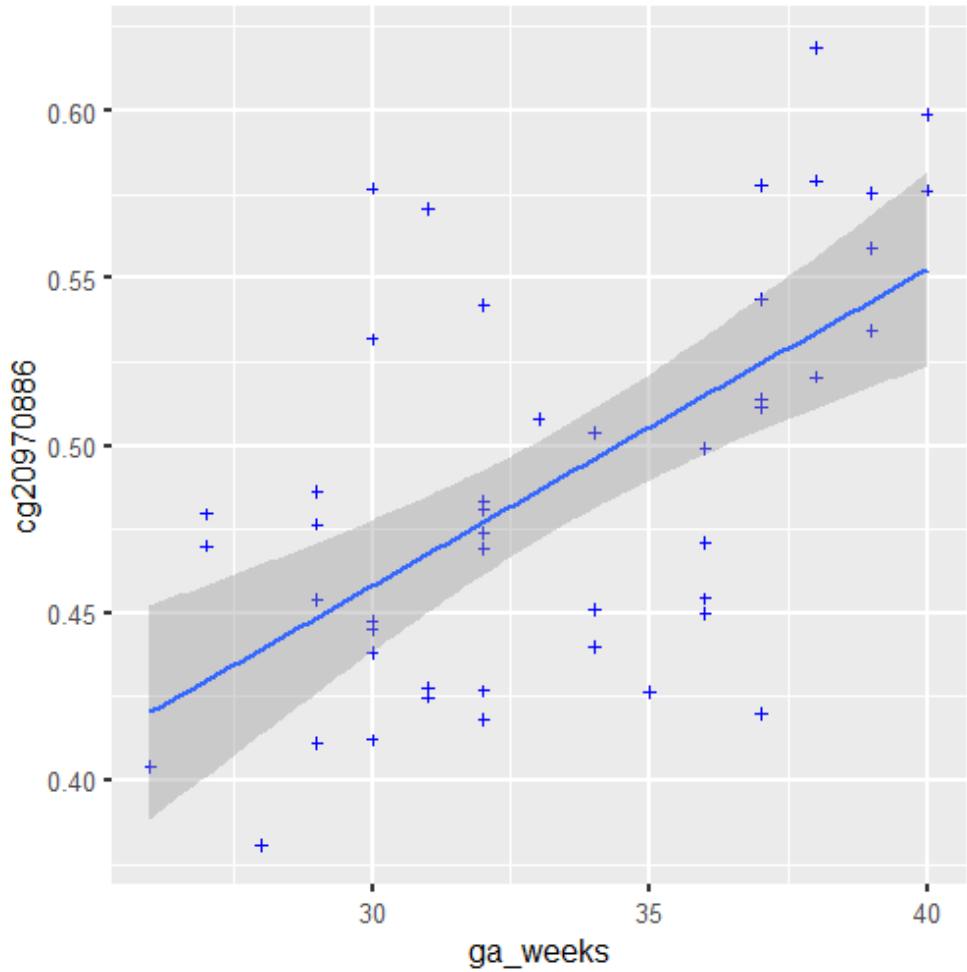
```
ggplot(data = geo_data,
       aes(x = ga_weeks,
           y = cg20970886)) +
  geom_point(aes(color = maternal_ethnicity),
             shape = 3) +
  geom_smooth(method = "lm")
```

See reference for complete list of shapes.



A common mistake is to forget the aesthetic call.

```
ggplot(data = geo_data,
       aes(x = ga_weeks,
           y = cg20970886)) +
  geom_point(color = "blue",
             shape = 3) +
  geom_smooth(method = "lm")
```



This assigns color to all the data

At this point, we've covered the 3 essential components to any ggplot2 plot:

1. **Data** - declare with a `ggplot(data = ...)` call
2. **Aesthetics** - assign input to plot components with `aes()`, e.g. (x/y position, color)
3. **Geoms** - declare the type of geometry, e.g. `+ geom_point()` for points

There are so many geoms

Each geom has their own required aesthetics, and optional ones

- `geom_point` requires `x` and `y`, and that they be numeric variables
- `geom_boxplot` requires `x` and `y`, but `x` must be categorical
- `geom_histogram` and `geom_density` requires `x`
- `geom_text` requires `x`, `y`, and `text`

Check out tidyverse site for full list.

You can visit help pages for more information on a specific geom's options (e.g. `?geom_point`)

Now we know the basics, we can explore ways to customize our plots

```
ggplot(data = geo_data)
```

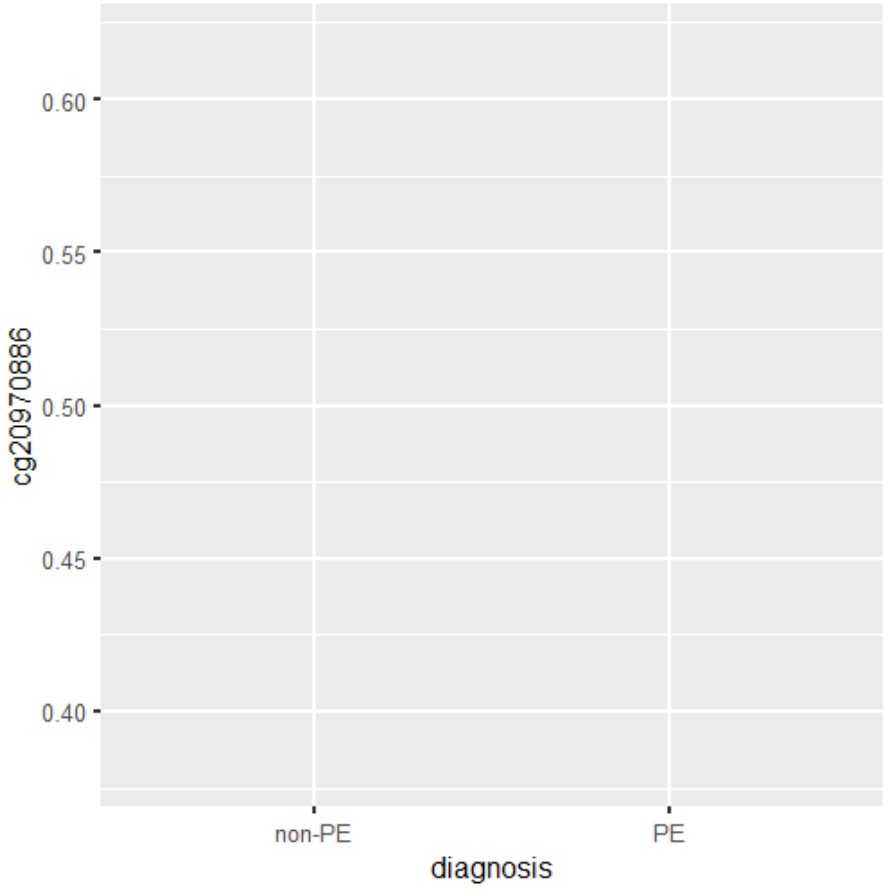
We'll start by looking at the methylation of this CpG site between preeclamptic and non-preeclamptic samples

First we declare the data.

PE: diagnosed with preeclampsia

```
ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis))
```

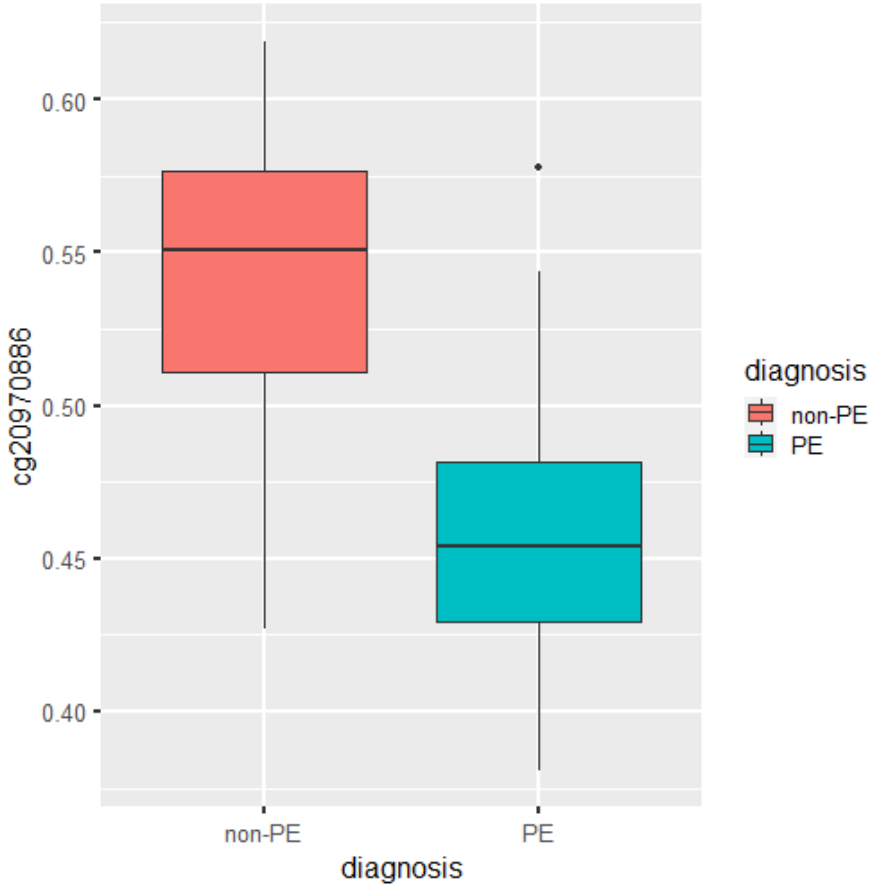
Then we declare the mappings of our variables to aesthetics



PE: diagnosed with preeclampsia

```
ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis)) +
  geom_boxplot()
```

To specify we want boxplots, we use `geom_boxplot`



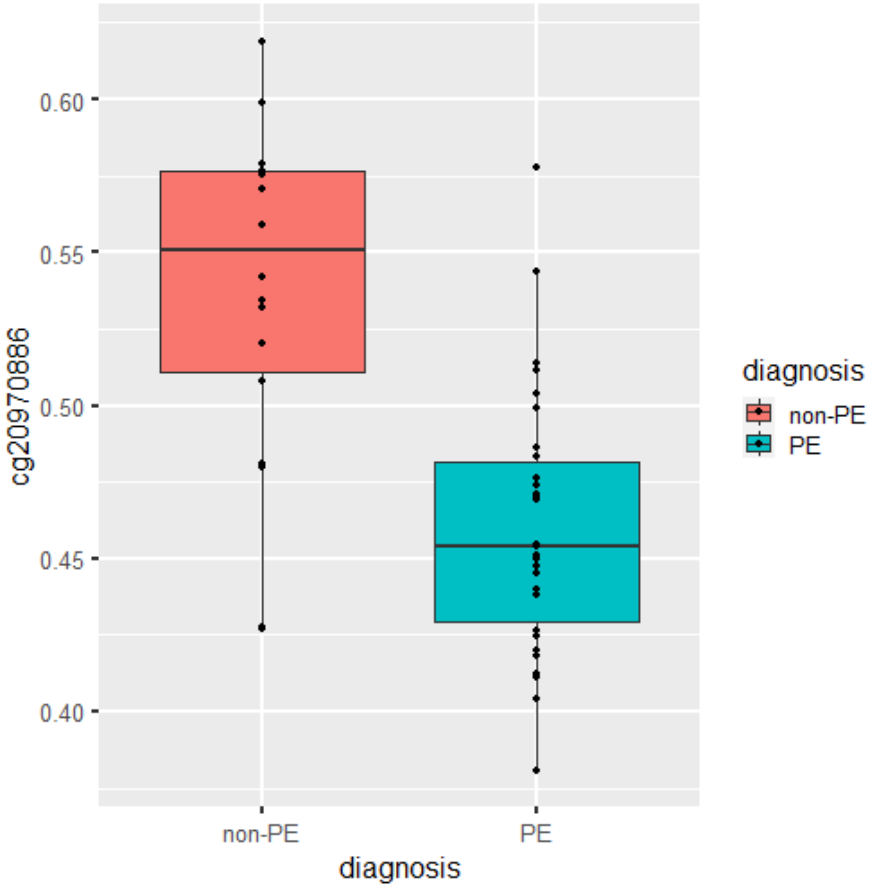
PE: diagnosed with preeclampsia

```
ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis)) +
  geom_boxplot() +
  geom_point()
```

It can be informative to plot all individual data points over top of the boxplots.

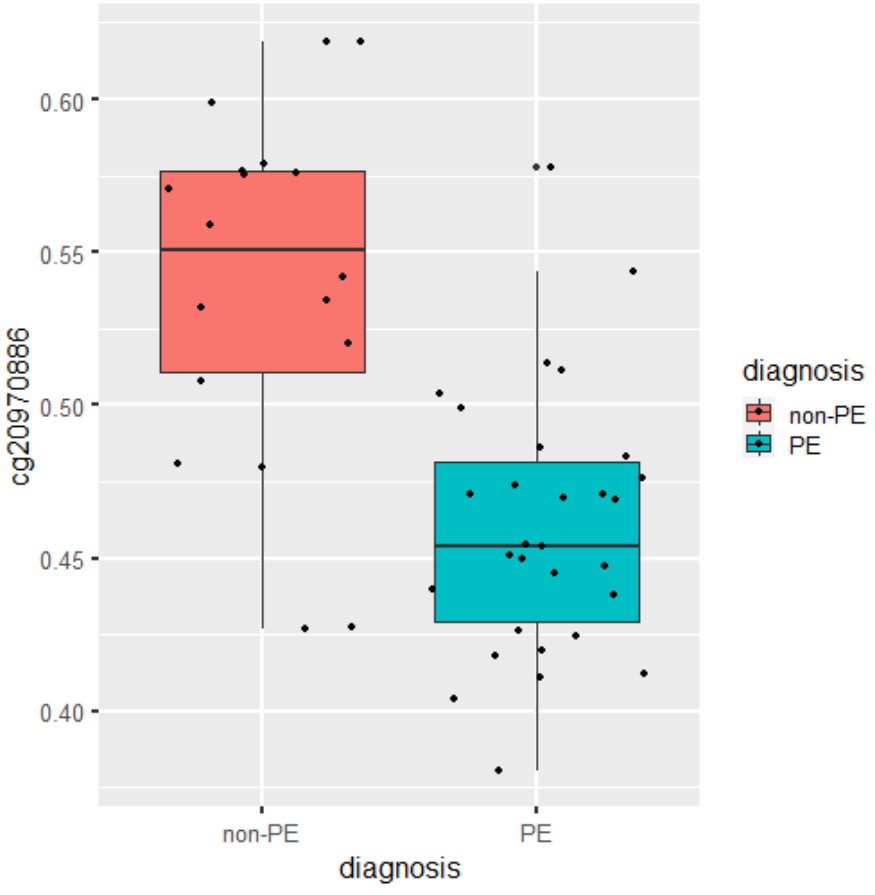
To add individual data points, we simply add another geometry, `geom_point`

But it's a bit hard to see when the points overlap each other..



```
ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis)) +
  geom_boxplot() +
  geom_jitter()
```

`geom_jitter` adds "noise" so that the points are spread out horizontally.



Customizing your graphs

Scales and themes

Scales

`aes` determines which data variables are mapped to each component of the graph

`scale_*_*` functions determine *how* this mapping is done

`scale_<aes>_<type>` calls all start with "`scale_`" followed by the target aesthetic (e.g. x, y, color, fill), and finished by the type (e.g. discrete, continuous).

For example,

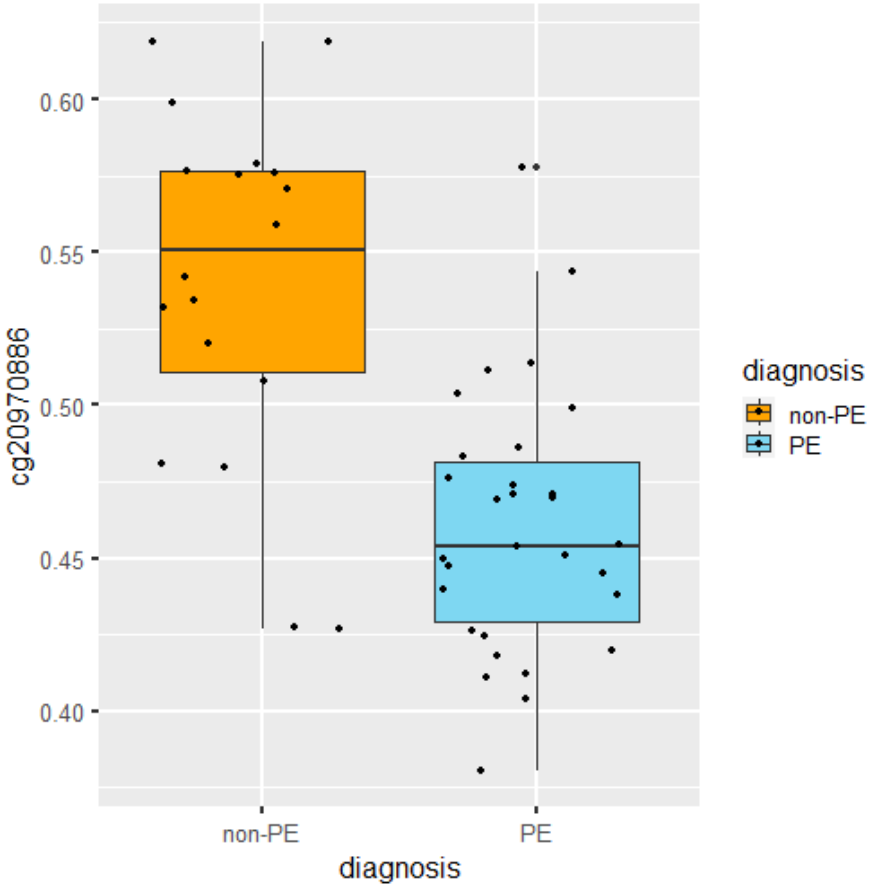
Want to change the limits on the y-axis? where the ticks appear? or maybe change to a log scale? Use `scale_y_continuous(limits = c(0,1))` or `scale_y_log10()`

Want to change colors? Use `scale_color_discrete()` for categorical variables `scale_color_continuous()` for continuous variables

```
ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis)) +
  geom_boxplot() +
  geom_jitter() +
  scale_fill_manual(values = c("orange", "#7ED7F2"))
```

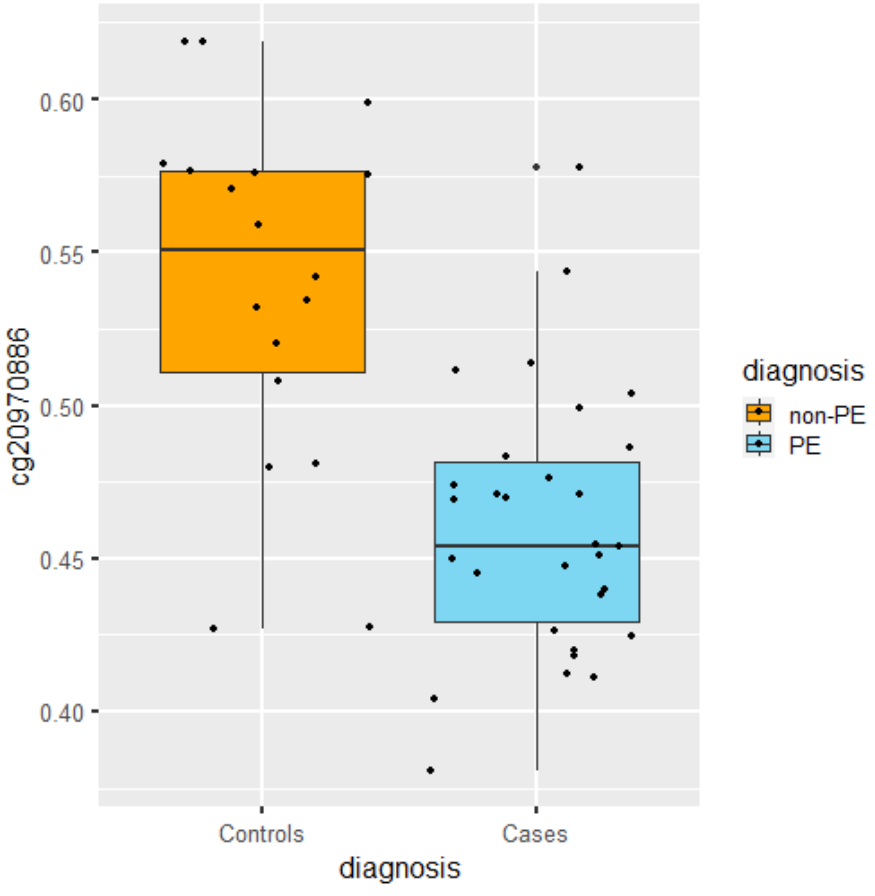
Here I assign specific colors to the categories of the diagnosis variable.

I supplied a vector of colors (can be in hex code) of same length of the number of categories of the variable `diagnosis`.



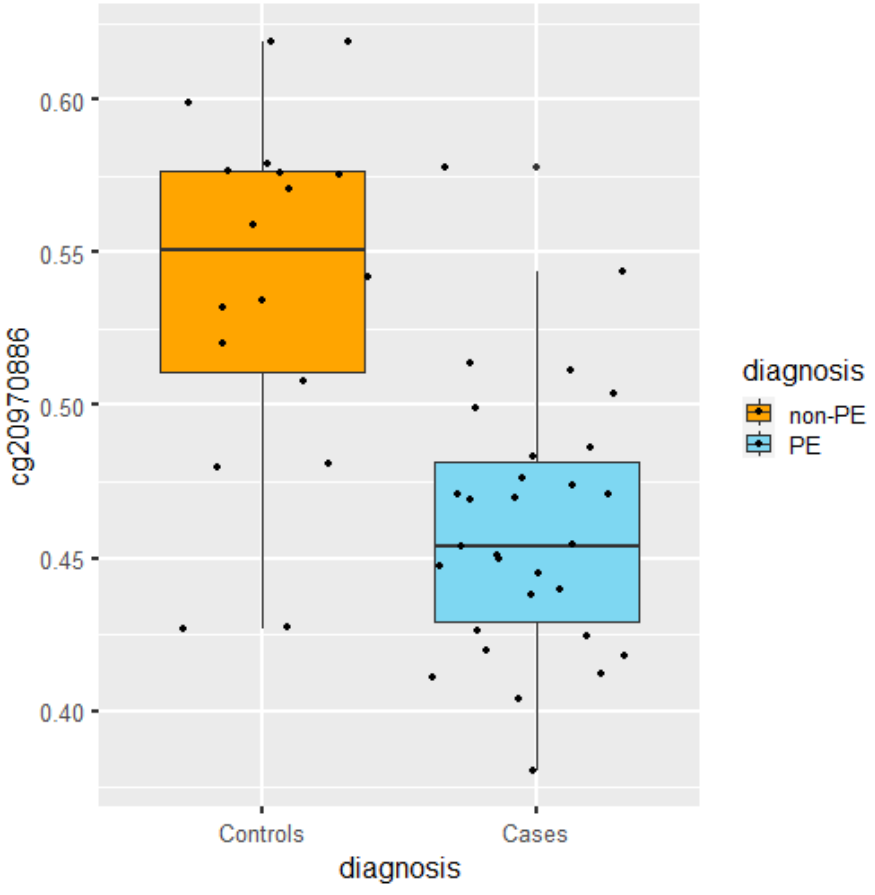
```
ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis)) +
  geom_boxplot() +
  geom_jitter() +
  scale_fill_manual(values = c("orange", "#7ED7F2")) +
  scale_x_discrete(labels = c("Controls",
                              "Cases"))
```

Here I change the labels of my x-axis.



```
ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis)) +
  geom_boxplot() +
  geom_jitter() +
  scale_fill_manual(values = c("orange", "#7ED7F2")) +
  scale_x_discrete(labels = c("non-PE" = "Controls",
                              "PE" = "Cases"))
```

It's better to be explicit about which label corresponds to which category

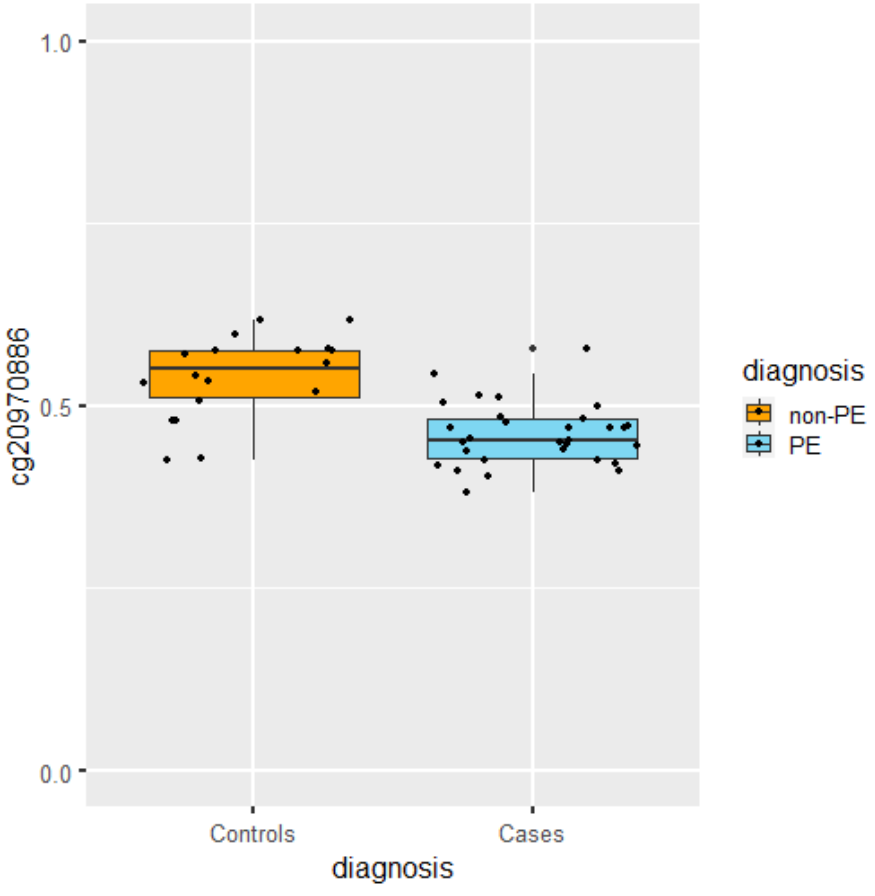


```r
ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis)) +
  geom_boxplot() +
  geom_jitter() +
  scale_fill_manual(values = c("orange", "#7ED7F2")) +
  scale_x_discrete(labels = c("non-PE" = "Controls",
                              "PE" = "Cases")) +
  scale_y_continuous(limits = c(0, 1),
                     breaks = c(0, 0.5, 1))
```

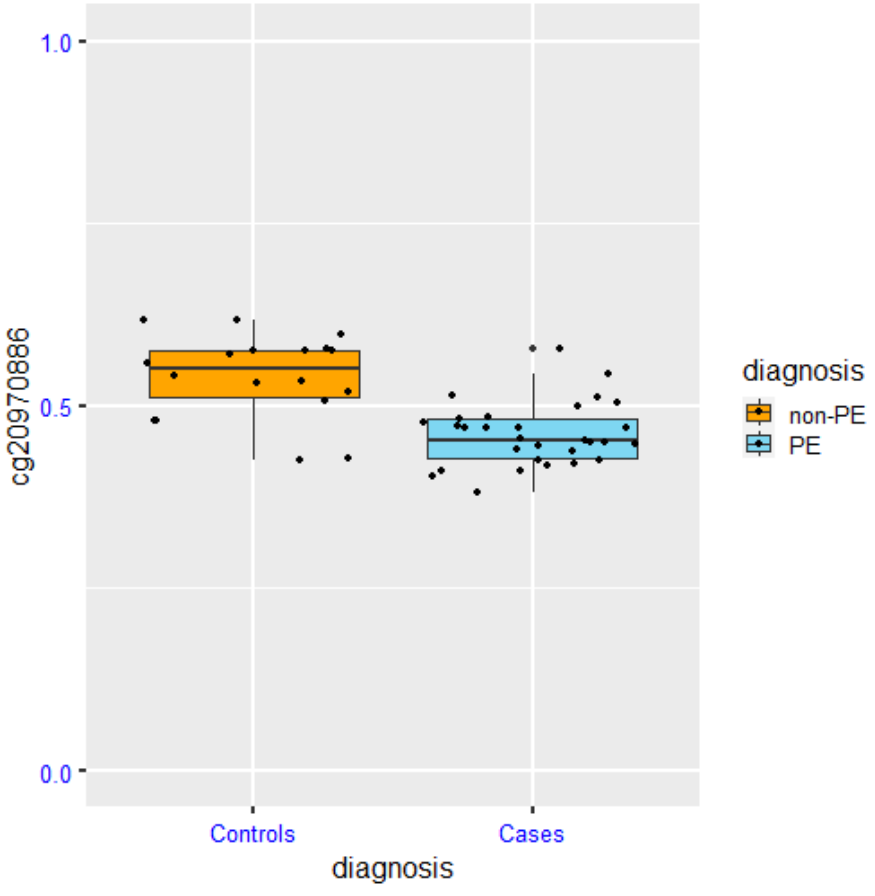
Here I expand the y axis to 0 and 1, the natural range of methylation.

I also change where I want the ticks (i.e. "breaks") to appear.

Note that the y axis is a numeric variable and x axis is categorical, and how the respective scale calls reflect that.



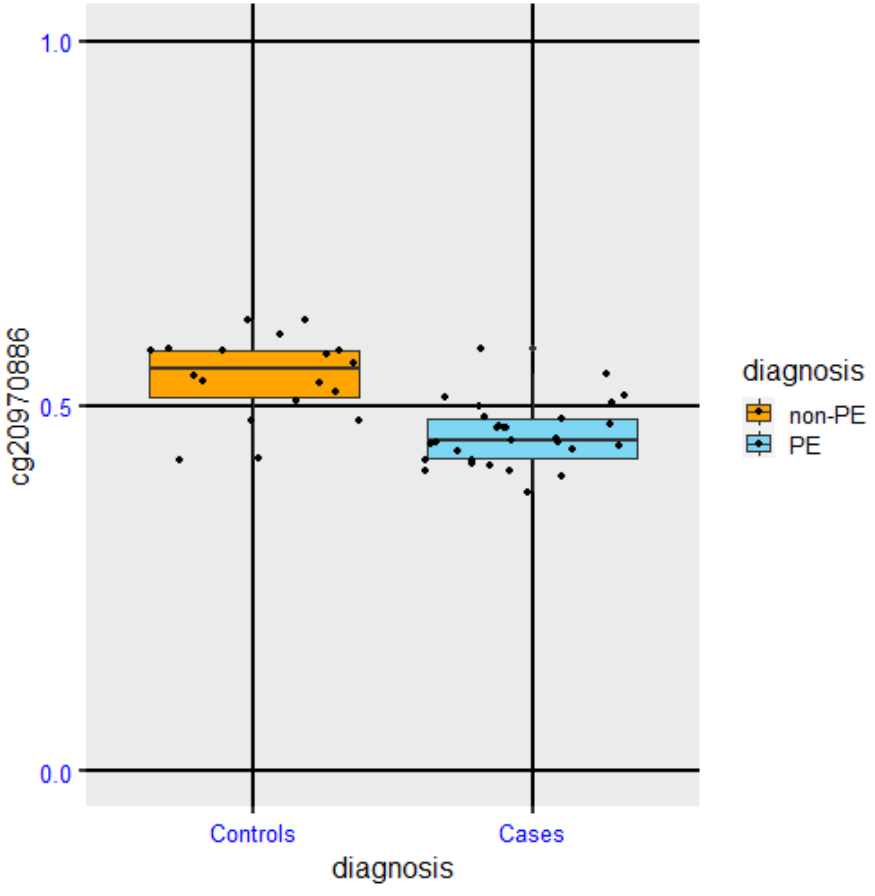
```
ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis)) +
  geom_boxplot() +
  geom_jitter() +
  scale_fill_manual(values = c("orange", "#7ED7F2")) +
  scale_x_discrete(labels = c("non-PE" = "Controls",
                              "PE" = "Cases")) +
  scale_y_continuous(limits = c(0, 1),
                     breaks = c(0, 0.5, 1))  +
  theme(axis.text = element_text(colour = 'blue'))
```



The **`theme()`** function call allows for a customization of the non-data components of a plot. Things like the title, labels, font size, gridlines, etc.

Pull up `?theme` to see a full description of all options

```
ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis)) +
  geom_boxplot() +
  geom_jitter() +
  scale_fill_manual(values = c("orange", "#7ED7F2")) +
  scale_x_discrete(labels = c("non-PE" = "Controls",
                              "PE" = "Cases")) +
  scale_y_continuous(limits = c(0, 1),
                     breaks = c(0, 0.5, 1))  +
  theme(axis.text = element_text(colour = 'blue'),
        panel.grid.major= element_line(colour = 'black'),
        panel.grid.minor = element_blank())
```

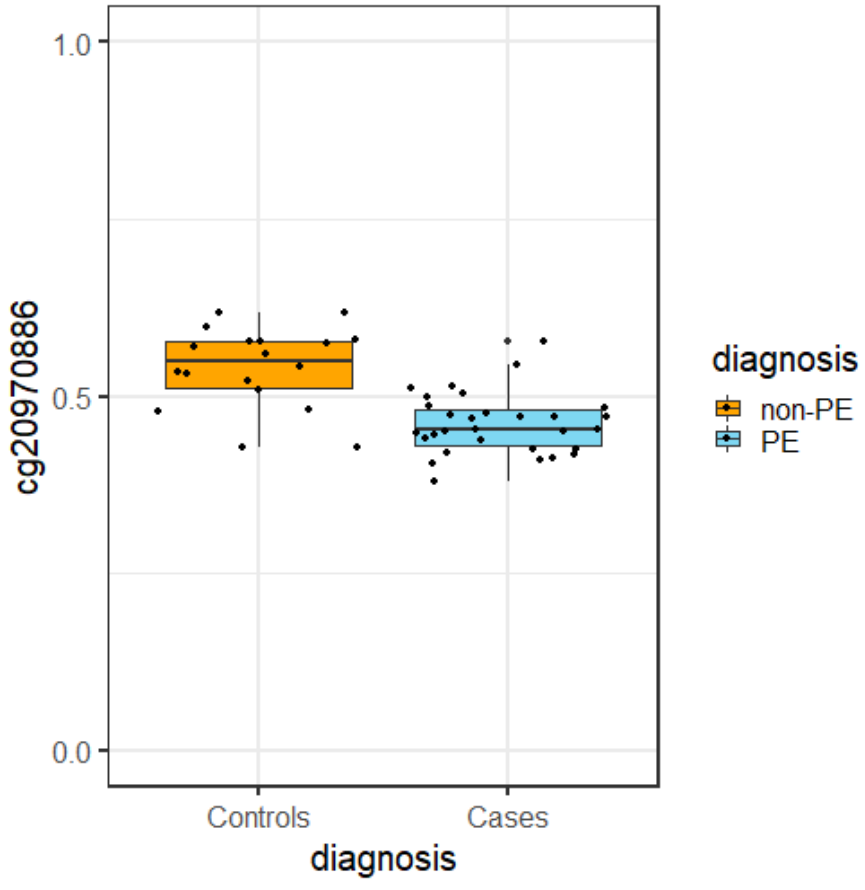


Most `theme()` arguments will require an "`element_*`" as input.

The type of element depends on the type of input (e.g. `element_text` for `axis.text`, `element_rect` for `panel.border`).

`element_blank` to remove components.

```r
ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis)) +
  geom_boxplot() +
  geom_jitter() +
  scale_fill_manual(values = c("orange", "#7ED7F2")) +
  scale_x_discrete(labels = c("non-PE" = "Controls",
                              "PE" = "Cases")) +
  scale_y_continuous(limits = c(0, 1),
                     breaks = c(0, 0.5, 1))  +
  theme_bw(base_size = 20)
```



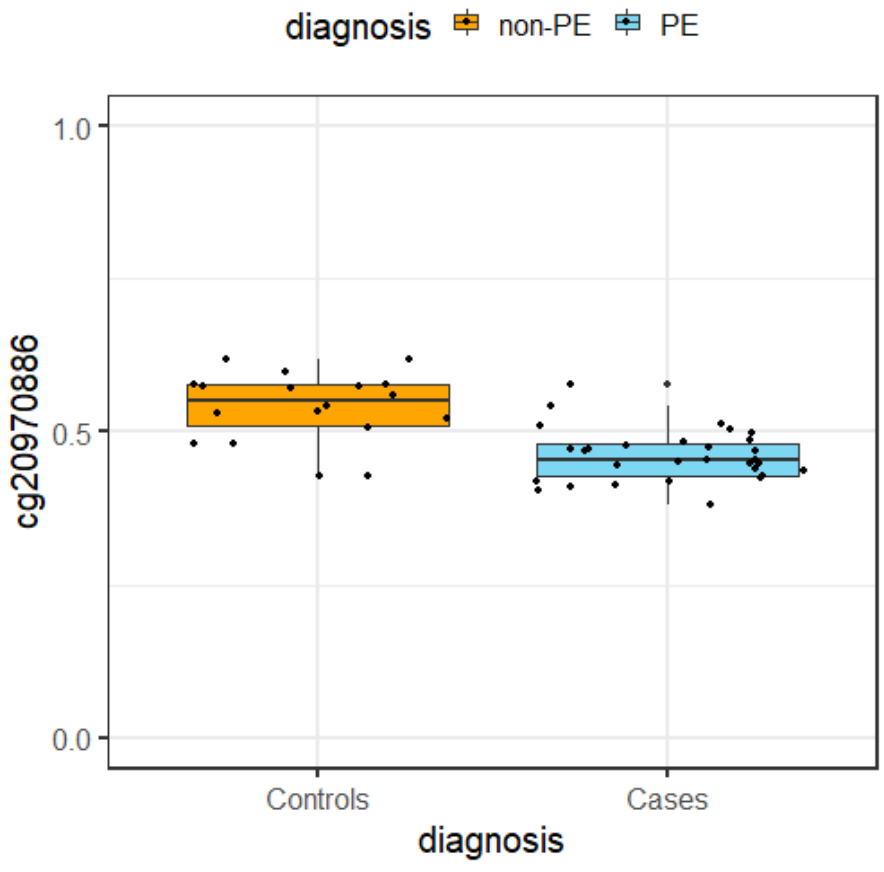
There are some predefined themes that look nice and easy to use.

- `theme_gray` - default ggplot2 theme
- `theme_classic` - minimal with no gridlines
- `theme_bw` - clean look with white background

List of complete ggplot2 themes

```
ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis)) +
  geom_boxplot() +
  geom_jitter() +
  scale_fill_manual(values = c("orange", "#7ED7F2")) +
  scale_x_discrete(labels = c("non-PE" = "Controls",
                              "PE" = "Cases")) +
  scale_y_continuous(limits = c(0, 1),
                     breaks = c(0, 0.5, 1))  +
  theme_bw(base_size = 20) +
  theme(legend.position = 'top')
```

You can customize these complete themes by calling `theme()` after e.g. `theme_bw()`



```
p <- ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis)) +
  geom_boxplot() +
  geom_jitter() +
  scale_fill_manual(values = c("orange", "#7ED7F2")) +
  scale_x_discrete(labels = c("non-PE" = "Controls",
                              "PE" = "Cases")) +
  scale_y_continuous(limits = c(0, 1),
                     breaks = c(0, 0.5, 1))  +
  theme_bw(base_size = 20) +
  theme(legend.position = 'top')
```

There are a couple of options to save plots in R.

Probably the simplest way is to use `ggsave` from `ggplot2`.

First thing to do is to assign your plot into an object.

I assigned our plot to the object named `p`

```r
p <- ggplot(data = geo_data,
       aes(x = diagnosis,
           y = cg20970886,
           fill = diagnosis)) +
  geom_boxplot() +
  geom_jitter() +
  scale_fill_manual(values = c("orange", "#7ED7F2")) +
  scale_x_discrete(labels = c("non-PE" = "Controls",
                              "PE" = "Cases")) +
  scale_y_continuous(limits = c(0, 1),
                     breaks = c(0, 0.5, 1))  +
  theme_bw(base_size = 20) +
  theme(legend.position = 'top')

ggsave(plot = p,
       filename = "this-plot.png",
       device = 'png',
       dpi = 72,
       height = 5,
       width = 7)
```

Then we can call `ggsave` on object `p`.

I would recommend specifying the following options:

- `filename`, the name and location where you want the plot to be saved
- `device`, the type of image file (e.g. "pdf", "png", "tiff", etc...)
- `height`, `width` - determines the dimensions of your plot
- `dpi`, resolution

After you run the code, check your local directory for the png file.

Resources

- Stack exchange for online help
- TOG study group / slack
- ggplot2 extensions
- ggplot2 cheatsheet
- r 4 data science data visualization chapter
- Eva Maerey's ggplot2 grammar guide