

Data Science for Environmental Health



Data Visualization

Recap

- `pivot_longer()` helps us take our data from wide to long format
 - `names_to` = gives a new name to the pivoted columns
 - `values_to` = gives a new name to the values that used to be in those columns
- `pivot_wider()` helps us take our data from long to wide format
 - `names_from` specifies the old column name that contains the new column names
 - `values_from` specifies the old column name that contains new cell values
- to merge/join data sets together need a variable in common - usually “id”

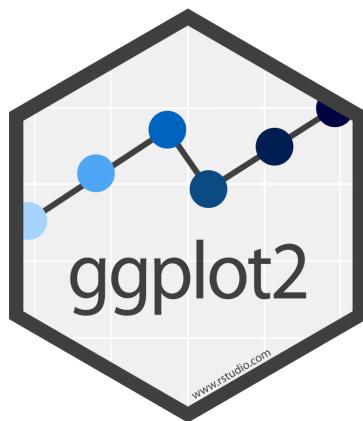
Cheatsheet

Recap continued

- to merge/join data sets together need a variable in common - usually "id"
- `?join` - see different types of joining for `dplyr`
- `inner_join(x, y)` - only rows that match for x and y are kept
- `full_join(x, y)` - all rows of x and y are kept
- `left_join(x, y)` - all rows of x are kept even if not merged with y
- `right_join(x, y)` - all rows of y are kept even if not merged with x
- `anti_join(x, y)` - all rows from x not in y keeping just columns from x.
- `esquisser()` function of the `esquisse` package can help make plot sketches

[Cheatsheet](#)

esquisse and ggplot2



Why learn ggplot2?

More customization:

- branding
- making plots interactive
- combining plots

Easier plot automation (creating plots in scripts)

Faster (eventually)

ggplot2

- A package for producing graphics - gg = *Grammar of Graphics*
- Created by Hadley Wickham in 2005
- Belongs to “Tidyverse” family of packages
- “*Make a ggplot*” = Make a plot with the use of ggplot2 package

Resources:

- <https://ggplot2-book.org/>
- <https://www.opencasestudies.org/>

ggplot2

Based on the idea of:

layering

plot objects are placed on top of each other with +

+

ggplot2

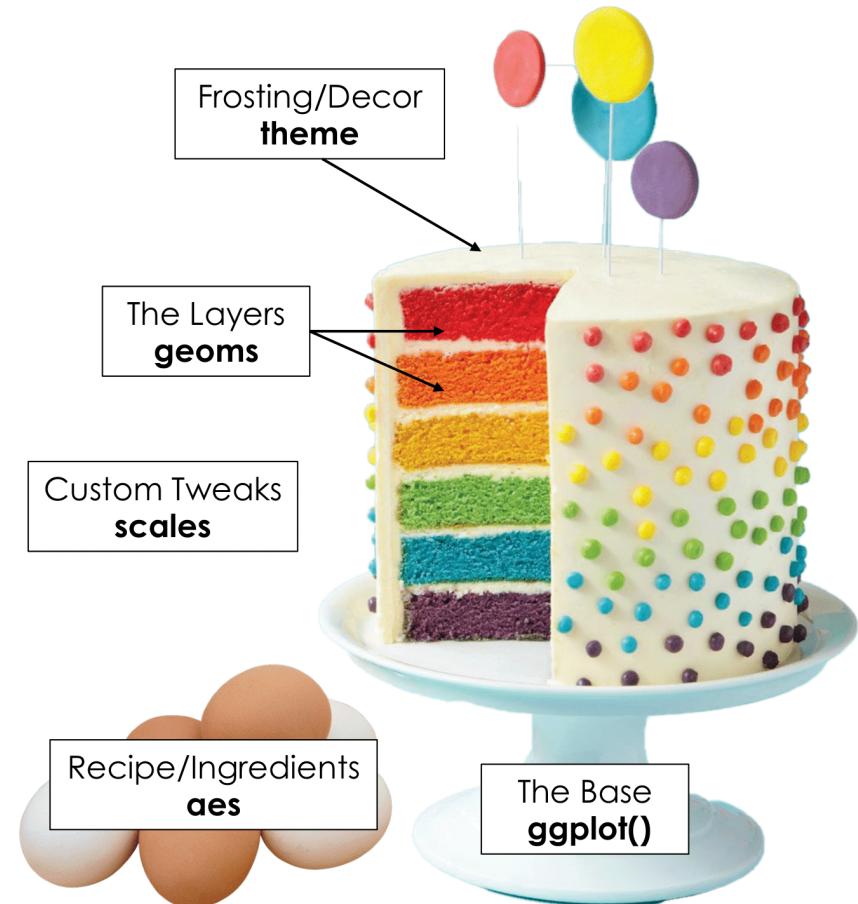
ggplot is a little bit like cake...

We always start by setting up the foundation with **ggplot()**

We specify our ingredients (data variables) with an **aes mapping**

We can create *layers* to our plot with **geoms**

We can style our *cake* ggplot with **themes**. We have out-of-the-box options, or we can go totally custom!



Slide Credit: Tanya Shapiro

ggplot2

- Pros: extremely powerful/flexible – allows combining multiple plot elements together, allows high customization of a look, many resources online
- Cons: ggplot2-specific “grammar of graphic” of constructing a plot
- [ggplot2 gallery](#)

Tidy data

To make graphics using `ggplot2`, our data needs to be in a **tidy** format

Tidy data:

1. Each variable forms a column.
2. Each observation forms a row.

Messy data:

- Column headers are values, not variable names.
- Multiple variables are stored in one column.
- Variables are stored in both rows and columns.

Tidy data: example

Ideally we want each variable as a column and we want each observation in a row.

Column headers are values, not variable names:

religion	<\$10k	\$10-20k	\$20-30k	\$30-40k	\$40-50k	\$50-75k
Agnostic	27	34	60	81	76	137
Atheist	12	27	37	52	35	70
Buddhist	27	21	30	34	33	58
Catholic	418	617	732	670	638	1116
Don't know/refused	15	14	15	11	10	35
Evangelical Prot	575	869	1064	982	881	1486
Hindu	1	9	7	9	11	34
Historically Black Prot	228	244	236	238	197	223
Jehovah's Witness	20	27	24	24	21	30
Jewish	19	19	25	25	30	95

Table 4: The first ten rows of data on income and religion from the Pew Forum. Three columns, \$75-100k, \$100-150k and >150k, have been omitted

Now the data is “tidy” and in long format

religion	income	freq
Agnostic	<\$10k	27
Agnostic	\$10-20k	34
Agnostic	\$20-30k	60
Agnostic	\$30-40k	81
Agnostic	\$40-50k	76
Agnostic	\$50-75k	137
Agnostic	\$75-100k	122
Agnostic	\$100-150k	109
Agnostic	>150k	84
Agnostic	Don't know/refused	96

Read more about tidy data and see other examples: [Tidy Data](#) tutorial

Data to plot

Type `?er_co_statewide` for more information.

Is the data in tidy? Is it in long format?

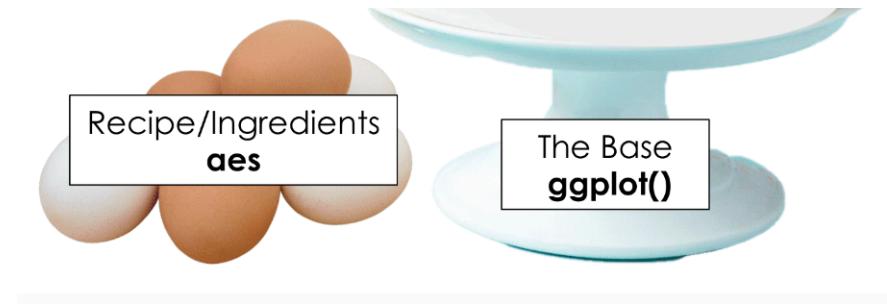
```
er_state <- er_CO_statewide  
head(er_state)
```

```
# A tibble: 6 × 5  
  rate lower95cl upper95cl visits year  
  <dbl>    <dbl>    <dbl>   <dbl> <dbl>  
1 6.51     5.80     7.23    323  2011  
2 6.58     5.88     7.29    339  2012  
3 5.82     5.16     6.49    302  2013  
4 4.44     3.87     5.01    237  2014  
5 6.55     5.86     7.25    355  2015  
6 8.46     7.68     9.23    467  2016
```

First plot with ggplot2 package

First layer of code with **ggplot2** package

Will set up the plot - it will be empty!

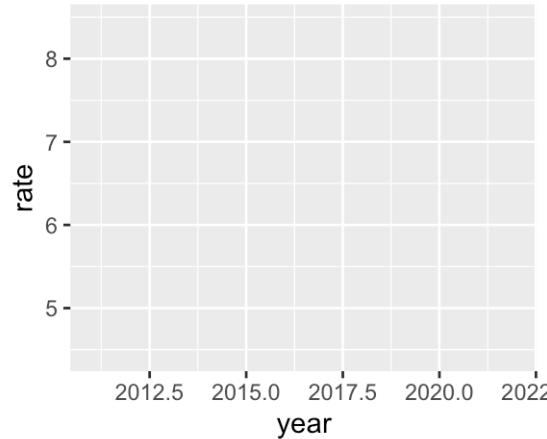


First layer of code with **ggplot2** package

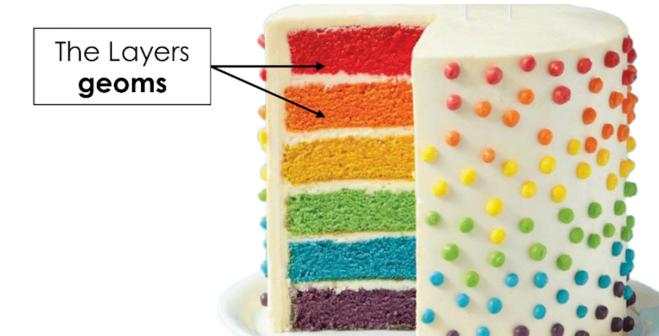
- **Aesthetic mapping** `aes(x= , y =)` describes how variables in our data are mapped to elements of the plot - Note you don't need to use `mapping` but it is helpful to know what we are doing.

```
library(ggplot2) # don't forget to load ggplot2  
# This is not code but shows the general format  
ggplot({data_to_plot}, aes(x = {var in data to plot},  
                           y = {var in data to plot}))
```

```
ggplot(er_state, aes(x = year, y = rate))
```



Next layer code with `ggplot2` package



There are many to choose from, to list just a few:

- `geom_point()` – points (we have seen)
- `geom_line()` – lines to connect observations
- `geom_boxplot()` – boxplots
- `geom_histogram()` – histogram
- `geom_bar()` – bar plot
- `geom_col()` – column plot
- `geom_tile()` – blocks filled with color

Next layer code with **ggplot2** package

When to use what plot? A few examples:

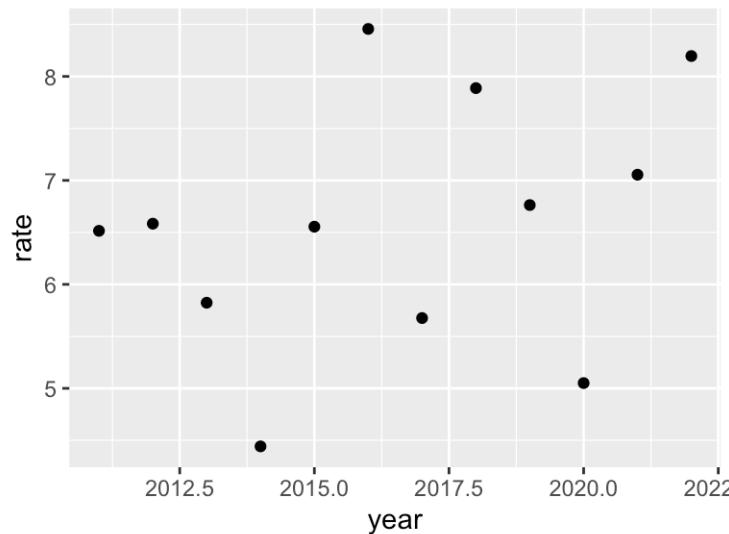
- a scatterplot (`geom_point()`): to examine the relationship between two sets of continuous numeric data
- a barplot (`geom_bar()`): to compare the distribution of a quantitative variable (numeric) between groups or categories
- a histogram (`geom_hist()`): to observe the overall distribution of numeric data
- a boxplot (`geom_boxplot()`): to compare values between different factor levels or categories

Next layer code with **ggplot2** package

Need the + sign to add the next layer to specify the type of plot

```
ggplot({data_to_plot}, aes(x = {var in data to plot},  
                           y = {var in data to plot})) +  
  geom_{type of plot}</div>
```

```
ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point()
```



Read as: *using CO statewide ER heat visits data, and provided aesthetic mapping, add points to the plot*

Tip - plus sign + must come at end of line

Having the + sign at the beginning of a line will not work!

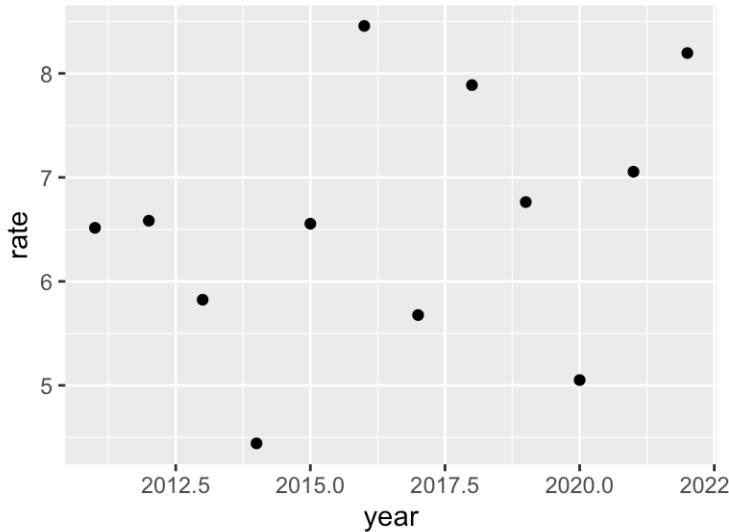
```
ggplot(er_state, aes(x = year,  
                      y = rate,  
                      fill = item_categ))  
+ geom_boxplot()
```

Pipes will also not work in place of +!

```
ggplot(er_state, aes(x = year,  
                      y = rate,  
                      fill = item_categ)) %>%  
geom_boxplot()
```

Plots can be assigned as an object

```
plt1 <- ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point()  
  
plt1
```

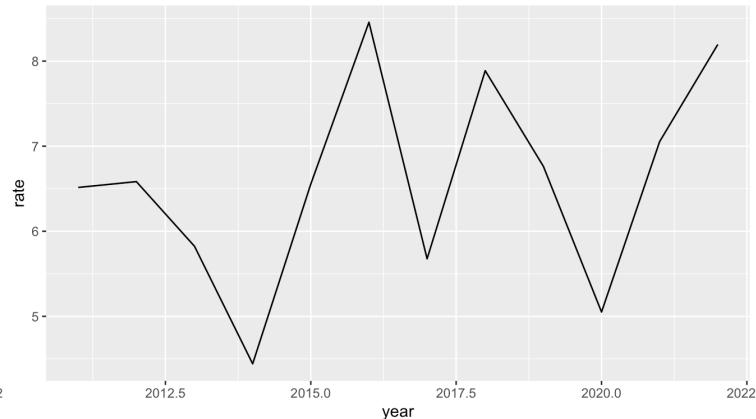
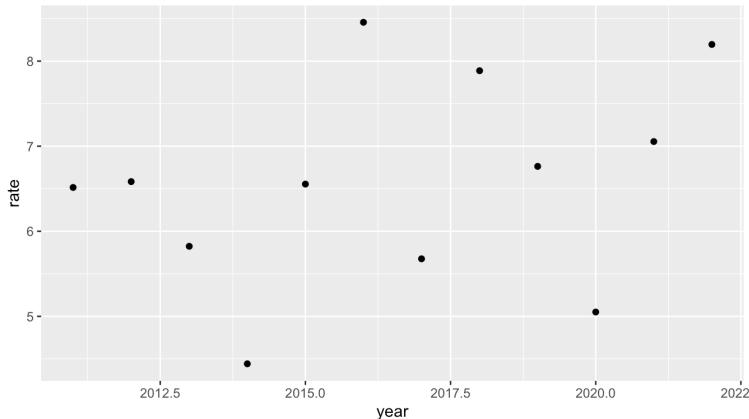


Examples of different geoms

```
plt1 <- ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point()
```

```
plt2 <- ggplot(er_state, aes(x = year, y = rate)) +  
  geom_line()
```

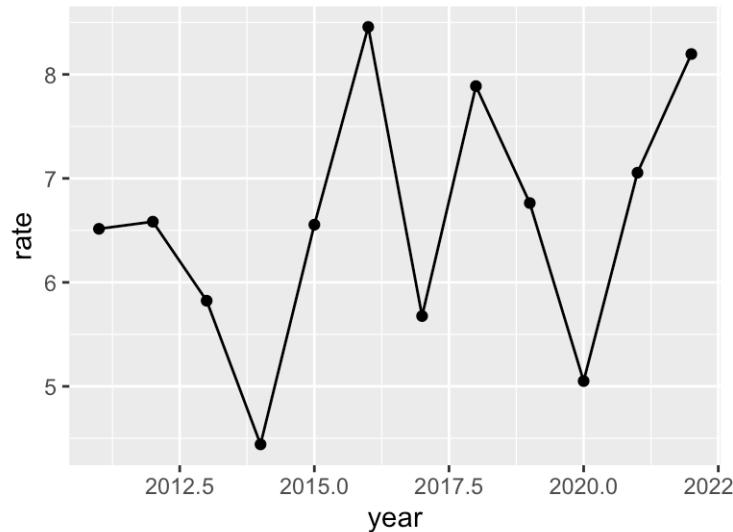
plt1 # *fig.show = "hold"* makes plots appear
plt2 # next to one another in the chunk settings



Specifying plot layers: combining multiple layers

Layer a plot on top of another plot with +

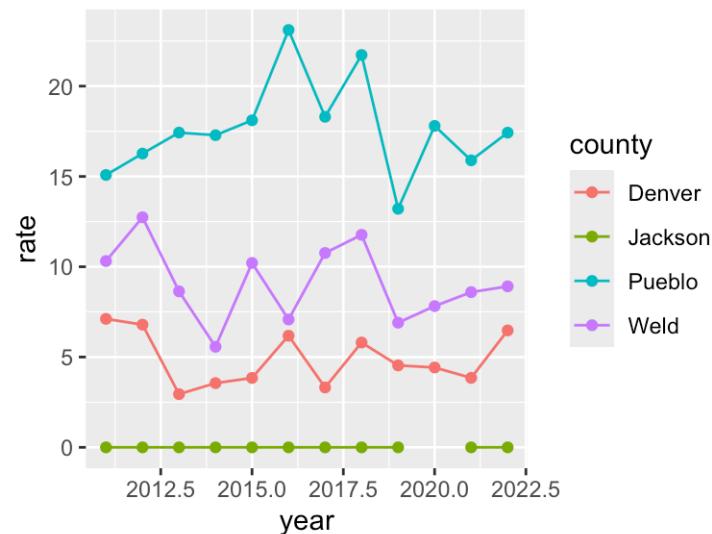
```
ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point() +  
  geom_line()
```



Adding color - can map color to a variable

```
set.seed(123)
er_visits_4 <- er_CO_county %>%
  filter(county %in% c("Denver", "Weld", "Pueblo", "Jackson"))

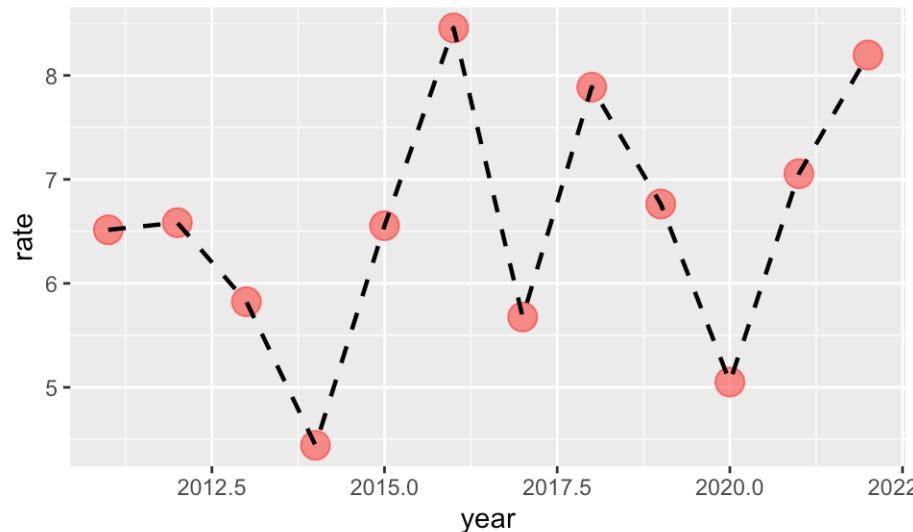
ggplot(er_visits_4, aes(x = year, y = rate, color = county)) +
  geom_point() +
  geom_line()
```



Adding color - or change the color of each plot layer

You can change look of each layer separately. Note the arguments like `linetype` and `alpha` that allow us to change the opacity of the points and style of the line respectively.

```
ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point(size = 5, color = "red", alpha = 0.5) +  
  geom_line(size = 0.8, color = "black", linetype = 2)
```



`linetype` can be given as a number. See the docs for what numbers correspond to what `linetype`!

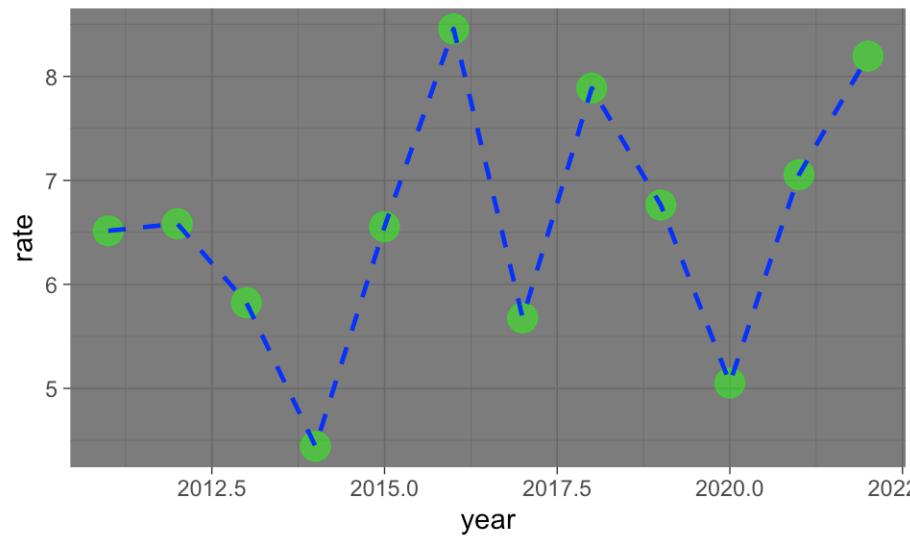


Customize the look of the plot

Customize the look of the plot

You can change the look of whole plot using [theme_*](#)() functions.

```
ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point(size = 5, color = "green", alpha = 0.5) +  
  geom_line(size = 0.8, color = "blue", linetype = 2) +  
  theme_dark()
```



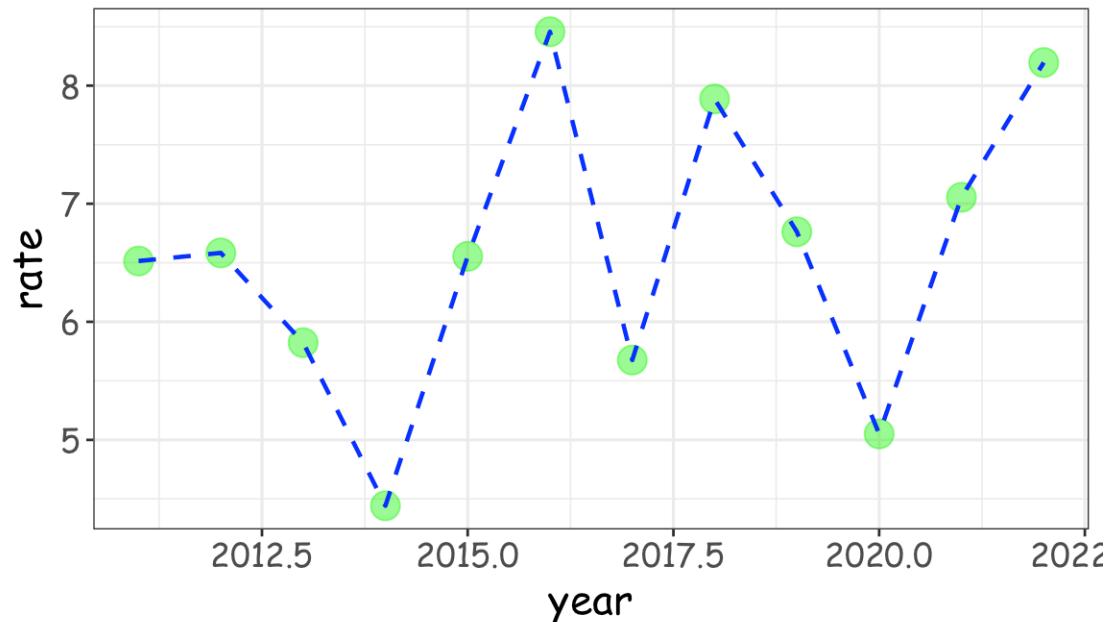
More themes!

There's not only the built in ggplot2 themes but all kinds of themes from other packages! - [ggthemes](#) - [ThemePark package](#) - [hrbr themes](#)

Customize the look of the plot

You can change the look of whole plot - **specific elements, too** - like changing font and font size - or even more [fonts](#)

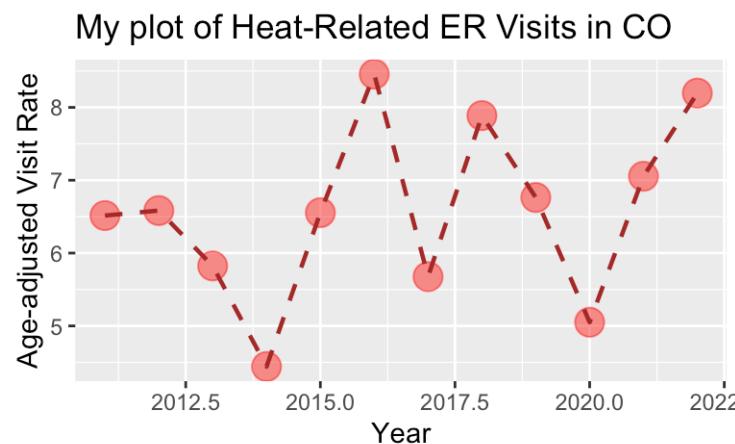
```
ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point(size = 5, color = "green", alpha = 0.5) +  
  geom_line(size = 0.8, color = "blue", linetype = 2) +  
  theme_bw() +  
  theme(text=element_text(size=16, family="Comic Sans MS"))
```



Adding labels

The `labs()` function can help you add or modify titles on your plot. The `title` argument specifies the title. The `x` argument specifies the x axis label. The `y` argument specifies the y axis label.

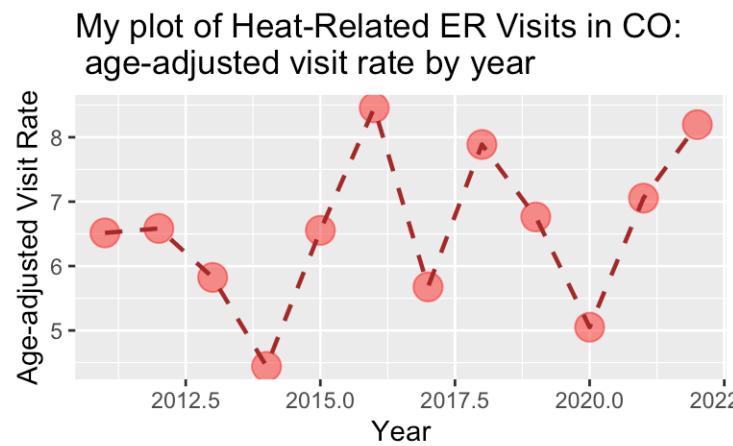
```
ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point(size = 5, color = "red", alpha = 0.5) +  
  geom_line(size = 0.8, color = "brown", linetype = 2) +  
  labs(title = "My plot of Heat-Related ER Visits in CO",  
       x = "Year",  
       y = "Age-adjusted Visit Rate")
```



Adding labels line break

Line breaks can be specified using `\n` within the `labs()` function to have a label with multiple lines.

```
ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point(size = 5, color = "red", alpha = 0.5) +  
  geom_line(size = 0.8, color = "brown", linetype = 2) +  
  labs(title = "My plot of Heat-Related ER Visits in CO: \n age-adjusted visit rate by year",  
       x = "Year",  
       y = "Age-adjusted Visit Rate")
```



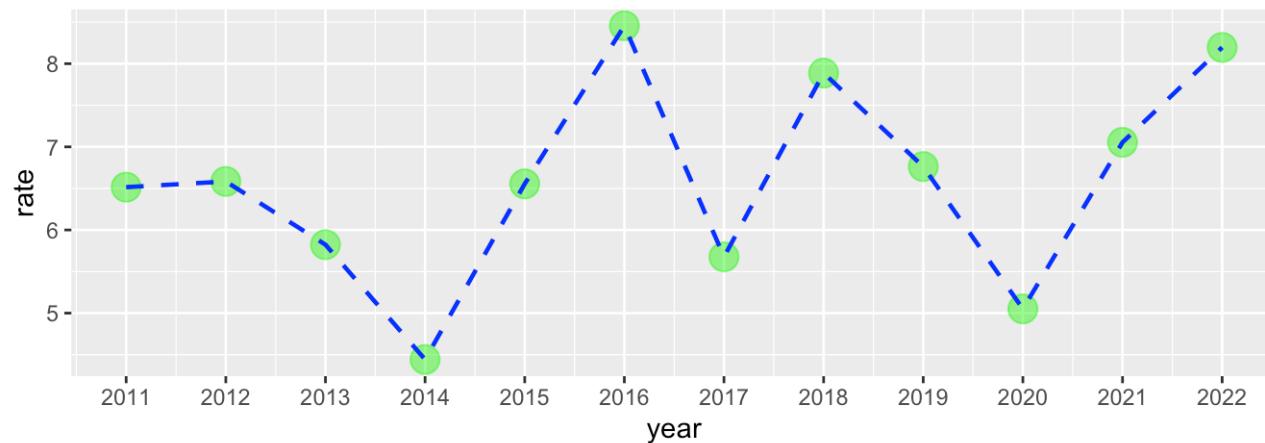
Changing axis: specifying axis scale

`scale_x_continuous()` and `scale_y_continuous()` can change how the axis is plotted. Can use the `breaks` argument to specify how you want the axis ticks.

```
range(pull(er_visits_4, year))
```

```
[1] 2011 2022
```

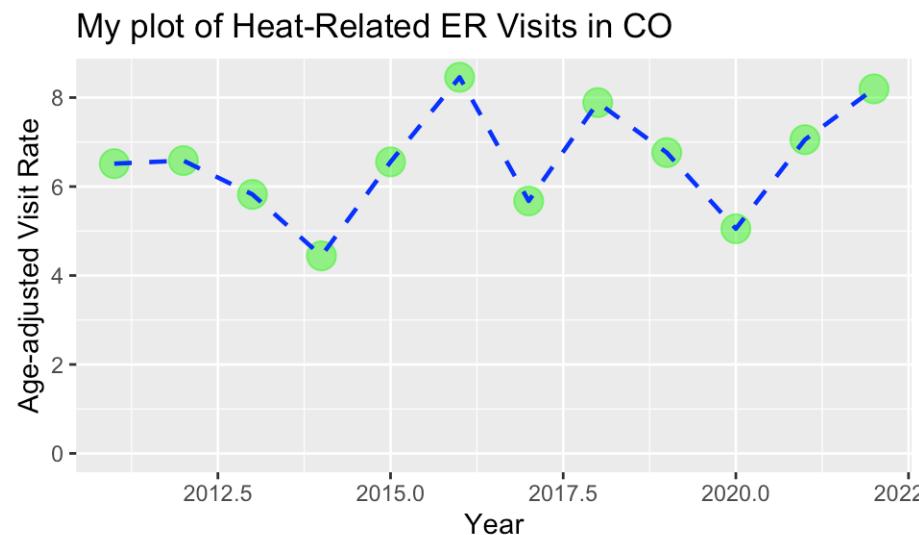
```
plot_scale <- ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point(size = 5, color = "green", alpha = 0.5) +  
  geom_line(size = 0.8, color = "blue", linetype = 2) +  
  scale_x_continuous(breaks = seq(from = 2011, to = 2022, by = 1))  
plot_scale
```



Changing axis: specifying axis limits

`xlim()` and `ylim()` can specify the limits for each axis

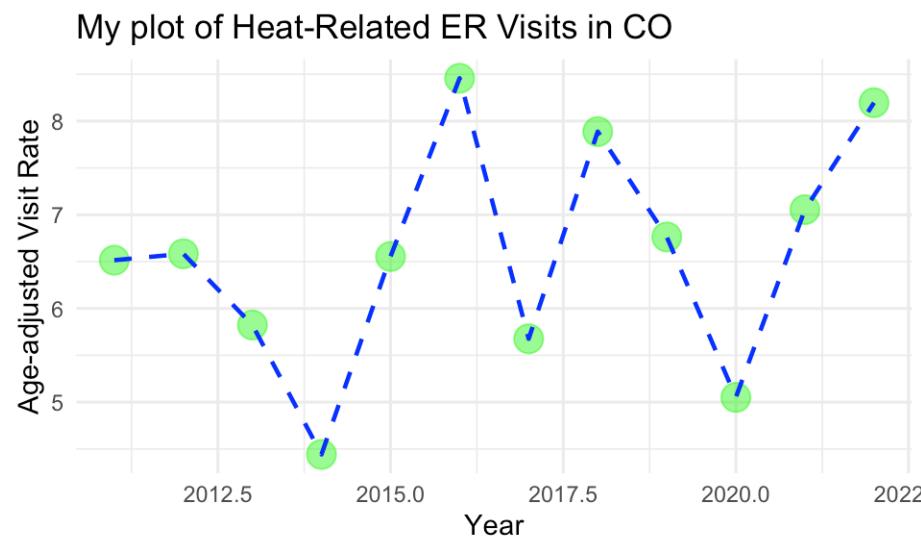
```
ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point(size = 5, color = "green", alpha = 0.5) +  
  geom_line(size = 0.8, color = "blue", linetype = 2) +  
  labs(title = "My plot of Heat-Related ER Visits in CO",  
       x = "Year",  
       y = "Age-adjusted Visit Rate") +  
  ylim(0, max(pull(er_visits_4, rate)))
```



Modifying plot objects

You can add to a plot object to make changes! Note that we can save our plots as an object like `plt1` below. And now if we reference `plt1` again our plot will print out!

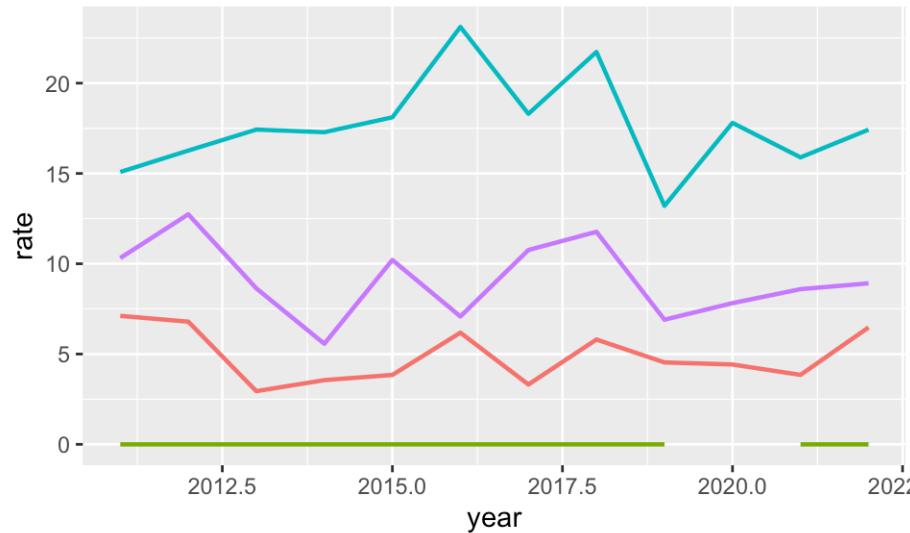
```
plt1 <- ggplot(er_state, aes(x = year, y = rate,)) +  
  geom_point(size = 5, color = "green", alpha = 0.5) +geom_line(size = 0.8, color = "blue", linetype = 2) +  
  labs(title = "My plot of Heat-Related ER Visits in CO", x = "Year", y = "Age-adjusted Visit Rate")  
  
plt1 + theme_minimal()
```



Removing the legend label

You can use `theme(legend.position = "none")` to remove the legend.

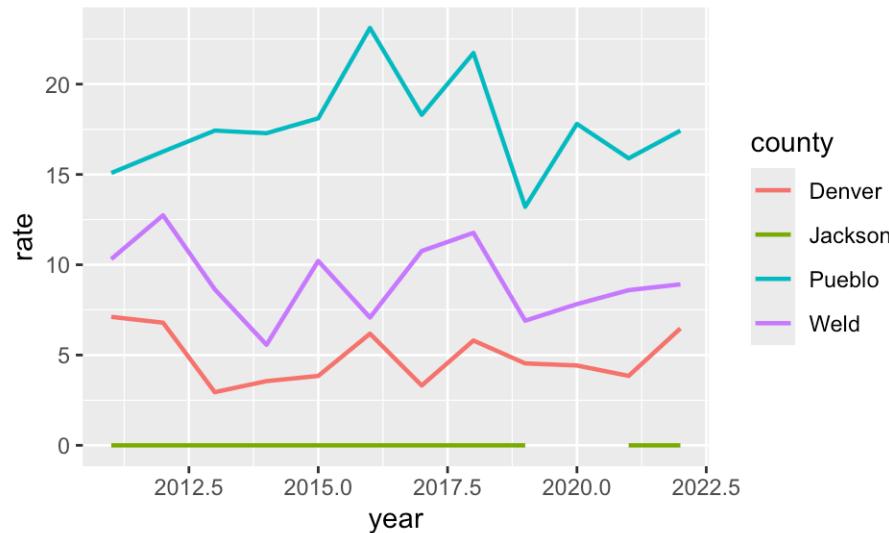
```
er_visits_4 %>% ggplot(aes(x = year,  
                               y = rate,  
                               color = county)) +  
  geom_line(size = 0.8) +  
  theme(legend.position = "none")
```



Overwriting specifications

It's possible to go in and change specifications with newer layers Here is our original plot.

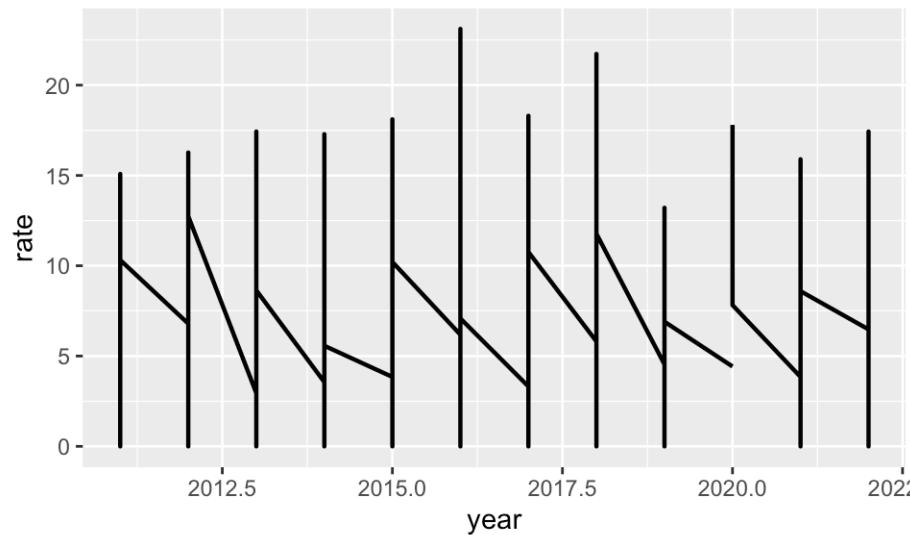
```
er_visits_4 %>% ggplot(aes(x = year,  
                               y = rate,  
                               color = county)) +  
  geom_line(size = 0.8)
```



Overwriting specifications

It's possible to go in and change specifications with newer layers

```
er_visits_4 %>% ggplot(aes(x = year,  
                               y = rate,  
                               color = county)) +  
  geom_line(size = 0.8, color = "black")
```



Summary

- `ggplot()` specifies what data use and what variables will be mapped to where
- inside `ggplot(), aes(x = , y = , color =)` specify what variables correspond to what aspects of the plot in general
- layers of plots can be combined using the `+` at the **end** of lines
- special theme_*() **functions** can change the overall look
- individual layers can be customized using arguments like: `size`, `color` `alpha` (more transparent is closer to 0), and `linetype`
- labels can be added with the `labs()` function and `x`, `y`, `title` arguments - the `\n` can be used for line breaks
- `xlim()` and `ylim()` can limit or expand the plot area
- `scale_x_continuous()` and `scale_y_continuous()` can modify the scale of the axes
- by default, `ggplot()` removes points with missing values from plots.

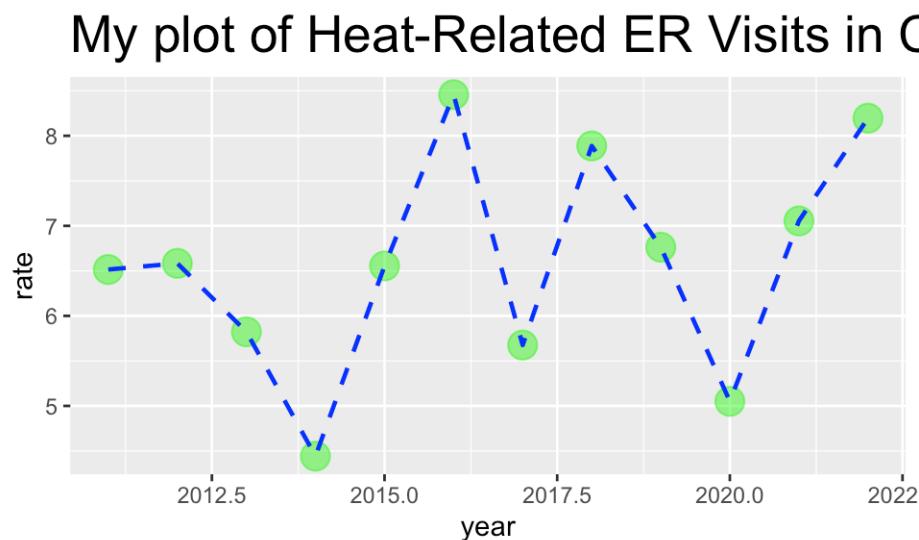
Lab 1

Class Website
Lab

theme() function:

The `theme()` function can help you modify various elements of your plot. Here we will adjust the font size of the plot title.

```
ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point(size = 5, color = "green", alpha = 0.5) +  
  geom_line(size = 0.8, color = "blue", linetype = 2) +  
  labs(title = "My plot of Heat-Related ER Visits in CO") +  
  theme(plot.title = element_text(size = 20))
```



theme() function

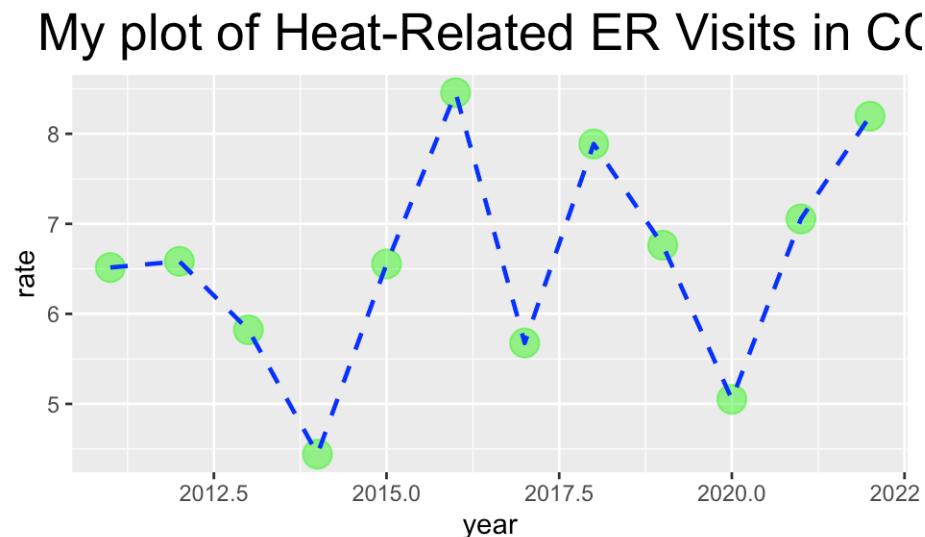
The `theme()` function always takes:

1. an object to change (use `?theme()` to see - `plot.title`, `axis.title`, `axis.ticks` etc.)
2. the aspect you are changing about this: `element_text()`, `element_line()`, `element_rect()`, `element_blank()`)
3. what you are changing:
 - `text`: `size`, `color`, `fill`, `face`, `alpha`, `angle`
 - `position`: "top", "bottom", "right", "left", "none"
 - `rectangle`: `size`, `color`, `fill`, `linetype`
 - `line`: `size`, `color`, `linetype`

theme() function: center title and change size

The `theme()` function can help you modify various elements of your plot. Here we will adjust the horizontal justification (`hjust`) of the plot title.

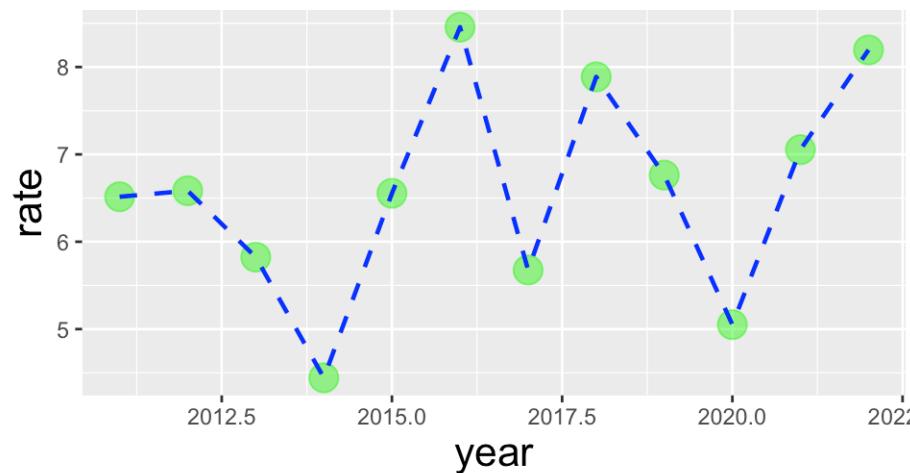
```
ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point(size = 5, color = "green", alpha = 0.5) +  
  geom_line(size = 0.8, color = "blue", linetype = 2) +  
  labs(title = "My plot of Heat-Related ER Visits in CO") +  
  theme(plot.title = element_text(hjust = 0.5, size = 20))
```



theme() function: change title and axis format

```
ggplot(er_state, aes(x = year, y = rate)) +  
  geom_point(size = 5, color = "green", alpha = 0.5) +  
  geom_line(size = 0.8, color = "blue", linetype = 2) +  
  labs(title = "My plot of Heat-Related ER Visits in CO") +  
  theme(plot.title = element_text(hjust = 0.5, size = 20),  
        axis.title = element_text(size = 16))
```

My plot of Heat-Related ER Visits in CO

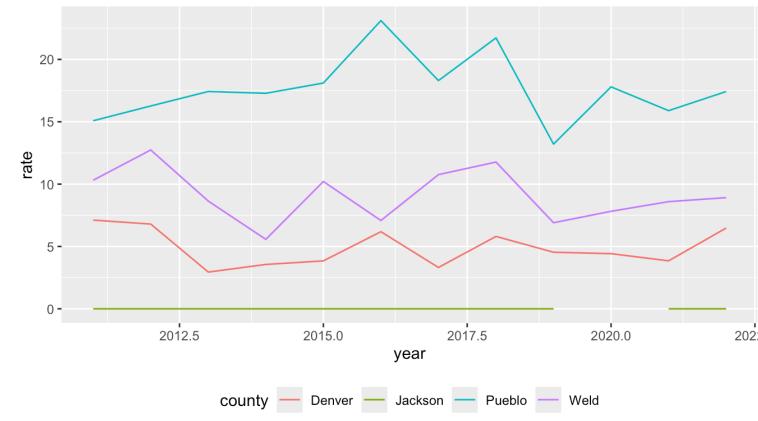
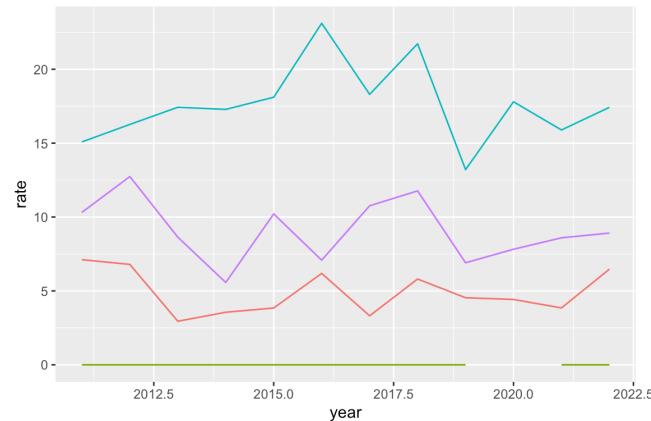


theme() function: moving (or removing) legend

If specifying position - use: "top", "bottom", "right", "left", "none"

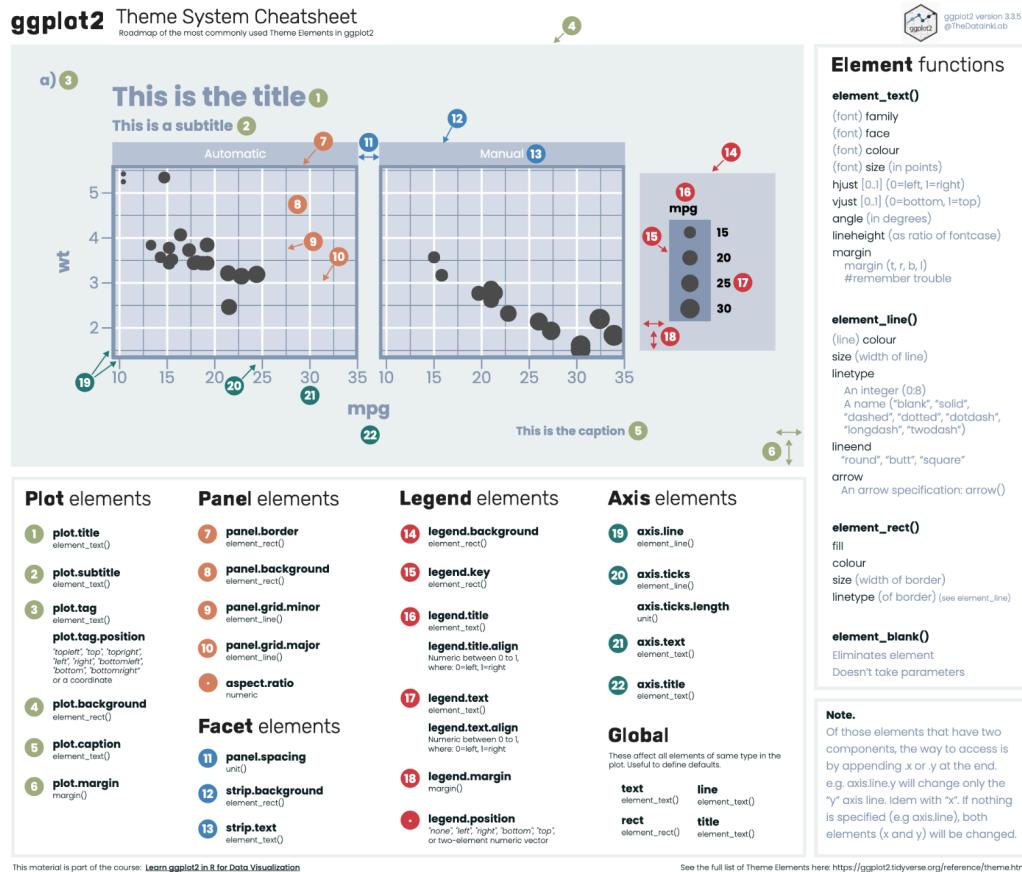
```
ggplot(er_visits_4, aes(x = year, y = rate, color = county)) +  
  geom_line()
```

```
ggplot(er_visits_4, aes(x = year, y = rate, color = county)) +  
  geom_line() +  
  theme(legend.position = "bottom")
```



Cheatsheet about theme

https://github.com/claragranell/ggplot2/blob/main/ggplot_theme_system_cheatsheet.pdf



Keys for specifications

linetype

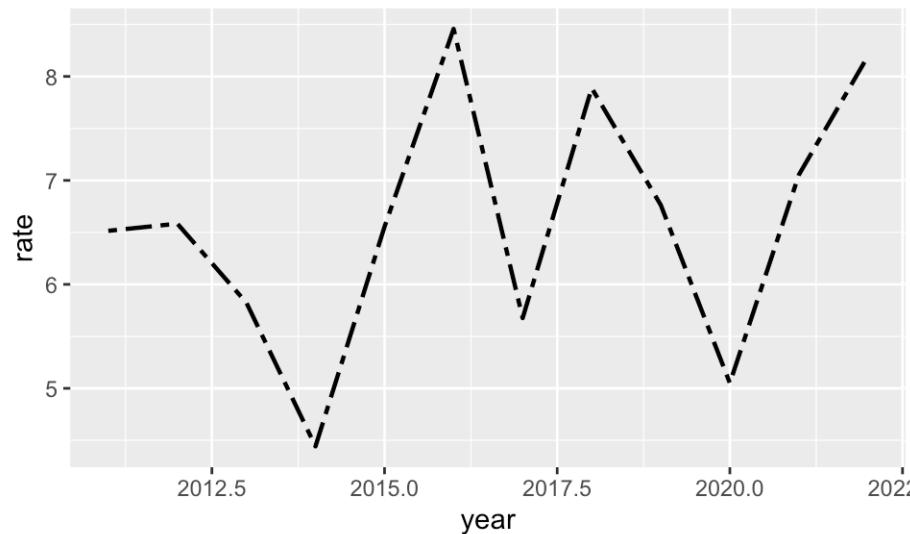
0. 'blank'	
1. 'solid'	<hr/>
2. 'dashed'	- - - - -
3. 'dotted'	· · · · ·
4. 'dotdash'	- · - · -
5. 'longdash'	- - - - -
6. 'twodash'	- - - - -

[source](#)

Linetype key

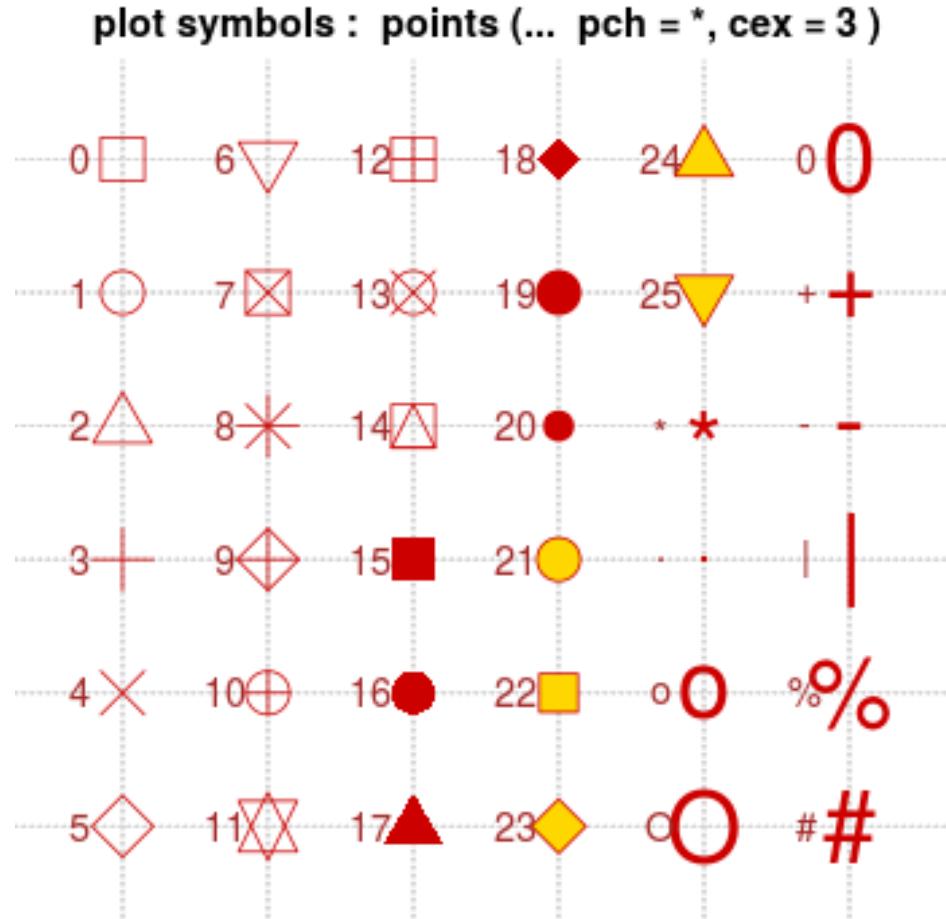
- *geoms* that draw lines have a `linetype` parameter
- these include values that are strings like “blank”, “solid”, “dashed”, “dotdash”, “longdash”, and “twodash”

```
er_state %>% ggplot(aes(x = year,  
                           y = rate)) +  
  geom_line(size = 0.8, linetype = "twodash")
```



Keys for specifications

shape

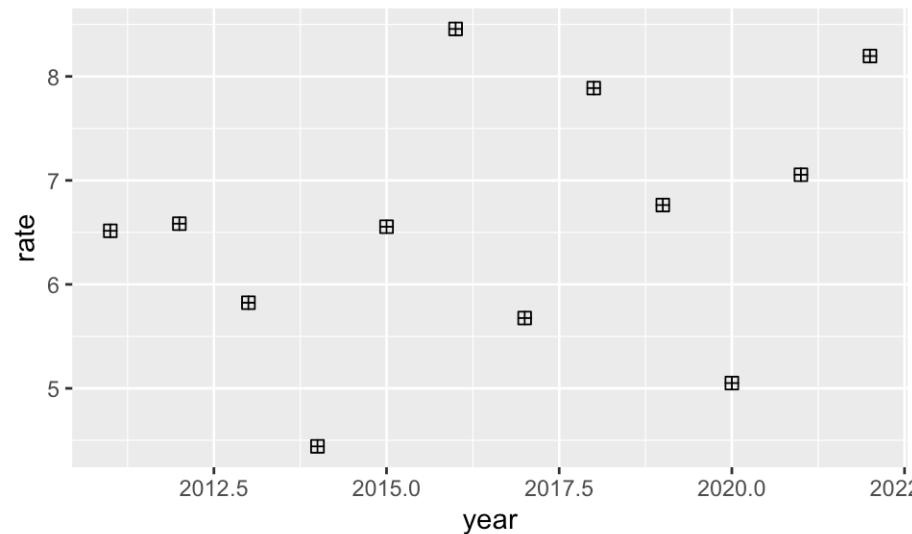


[source](#)

shape key

- *geoms* that draw have points have a shape parameter
- these include numeric values (don't need quotes for these) and some characters values (need quotes for these)

```
er_state %>% ggplot(aes(x = year,  
                           y = rate)) +  
  geom_point(size = 2, shape = 12)
```



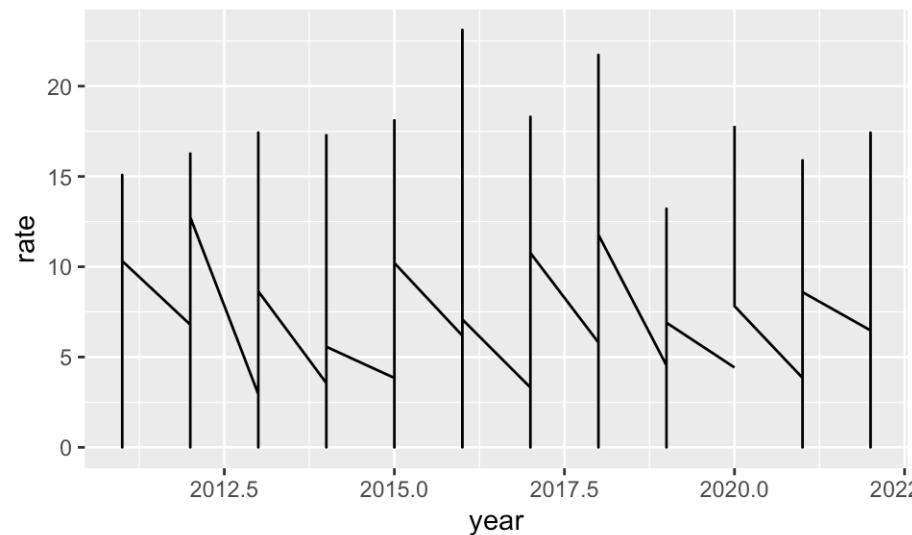
Can make your own theme to use on plots!

Guide on how to: <https://rpubs.com/mclaire19/ggplot2-custom-themes>

Starting a plot

Let's start with `er_visits_4`.

```
ggplot(er_visits_4, aes(x = year,  
                         y = rate)) +  
  geom_line()
```



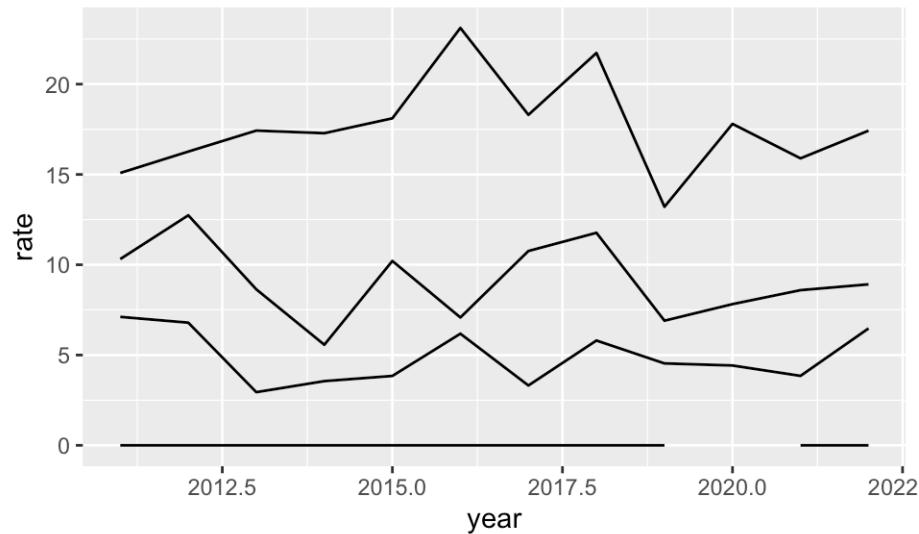
If it looks confusing to you, try again



Using group in plots

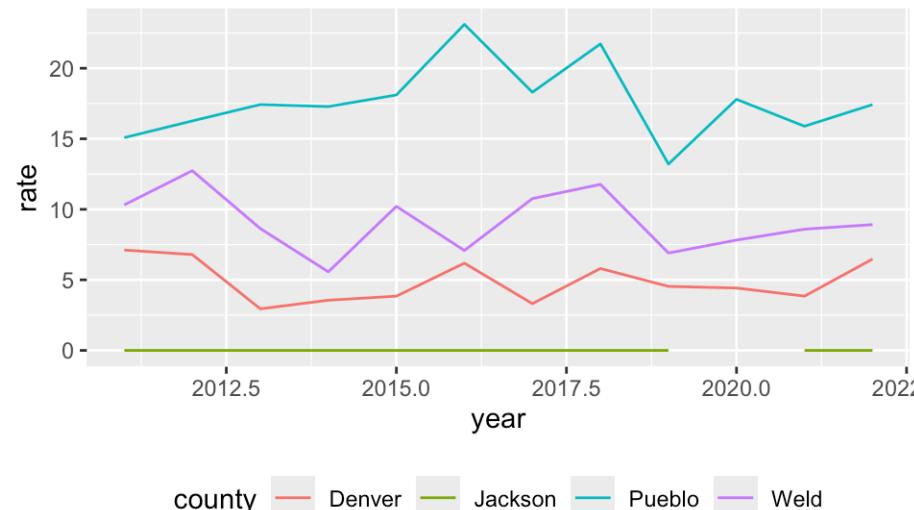
You can use `group` element in a mapping to indicate that each county will have a rate line.

```
ggplot(er_visits_4, aes(x = year,  
                         y = rate,  
                         group = county)) +  
  geom_line()
```



Adding color will automatically group the data

```
ggplot(er_visits_4, aes(x = year,  
                         y = rate,  
                         color = county)) +  
  geom_line() +  
  theme(legend.position = "bottom")
```



Tips!

Let's talk additional tricks and tips for making ggplots!

We are going to use some other data about ER visits that has to do with gender. Note that gender was recorded as binary, which we know isn't really accurate. This is something you might encounter. Please see this article about ways to measure gender in a more inclusive way:

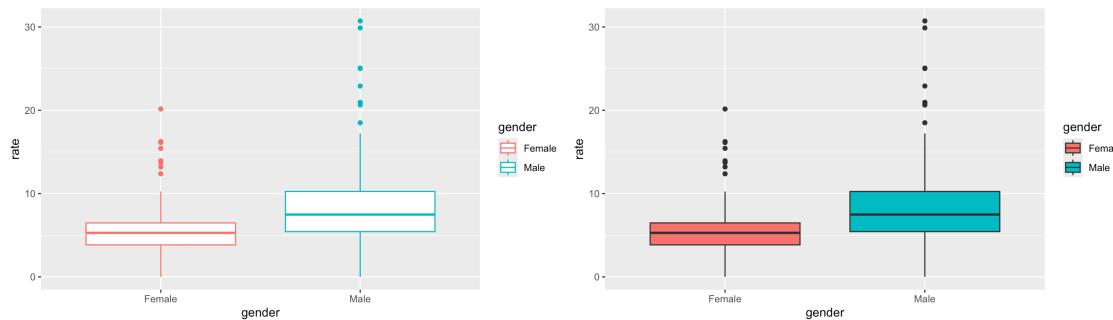
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6526522/>.

Tips - Color vs Fill

- `color` is needed for points and lines
- `fill` is generally needed for boxes and bars

```
er_visits_gender <- CO_heat_ER_bygender  
ggplot(er_visits_gender, aes(x = gender,  
                               y = rate,  
                               color = gender)) + #color creates an outline  
geom_boxplot()
```

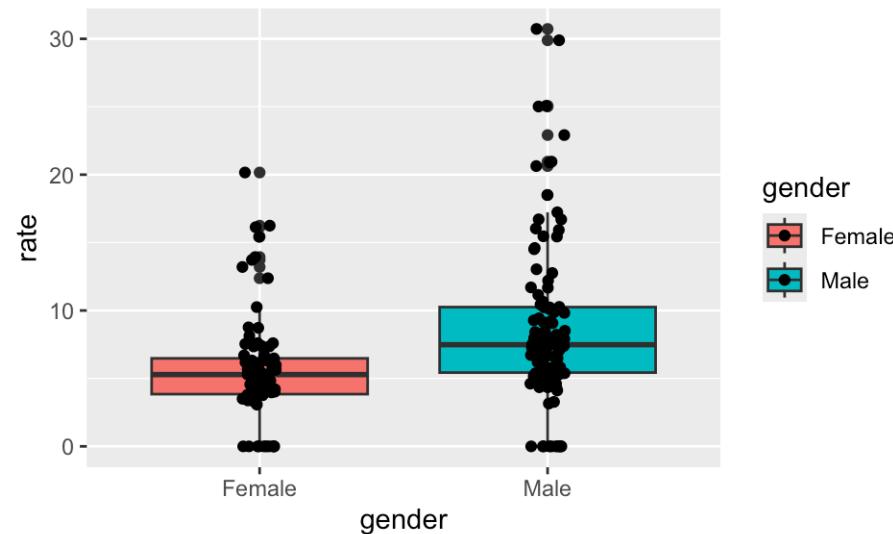
```
ggplot(er_visits_gender, aes(x = gender,  
                               y = rate,  
                               fill = gender)) + # fills the boxplot  
geom_boxplot()
```



Tip - Good idea to add jitter layer to top of box plots

Can add `width` argument to make the jitter more narrow.

```
ggplot(er_visits_gender, aes(x = gender,  
                               y = rate,  
                               fill = gender)) +  
  geom_boxplot() +  
  geom_jitter(width = .06)
```

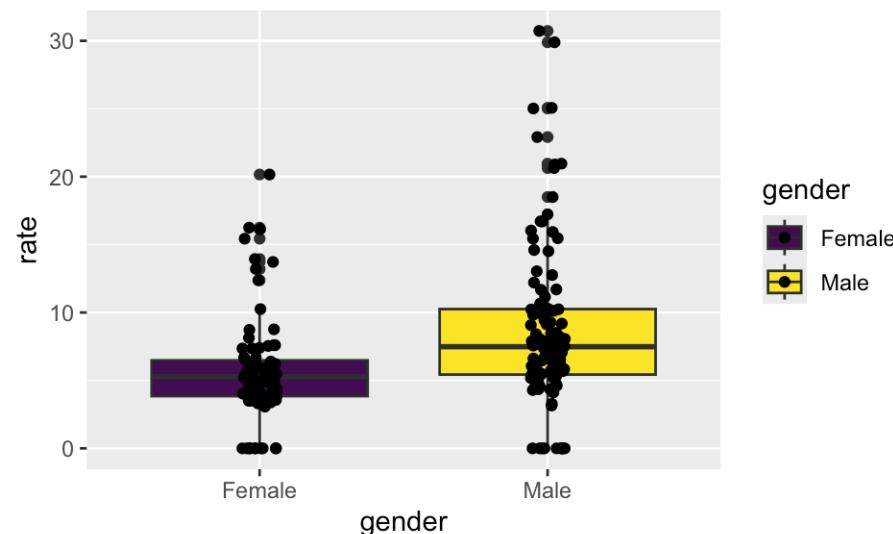


Tip - be careful about colors for color vision deficiency

`scale_fill_viridis_d()` for discrete /categorical data

`scale_fill_viridis_c()` for continuous data

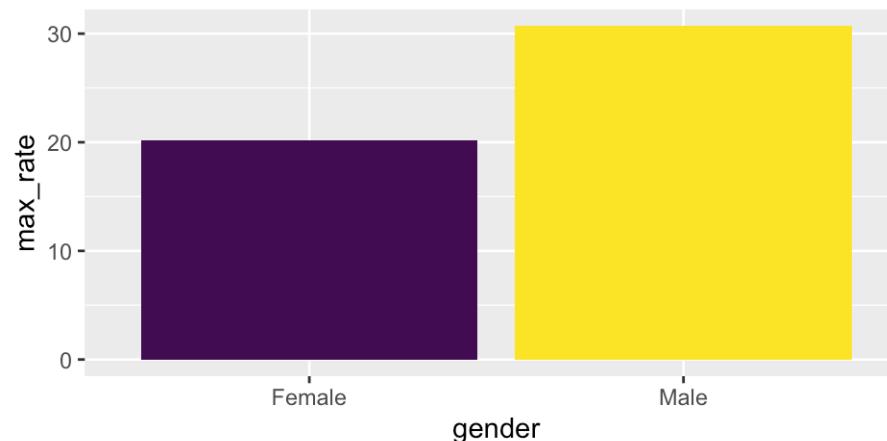
```
ggplot(er_visits_gender, aes(x = gender,  
                               y = rate,  
                               fill = gender)) +  
  geom_boxplot() +  
  geom_jitter(width = .06) +  
  scale_fill_viridis_d()
```



Tip - can pipe data after wrangling into ggplot()

```
er_bar <- er_visits_gender %>%
  group_by(gender) %>%
  summarize("max_rate" = max(rate, na.rm=T)) %>%
  
  ggplot(aes(x = gender,
              y = max_rate,
              fill = gender)) +
  scale_fill_viridis_d()+
  geom_col() +
  theme(legend.position = "none")
```

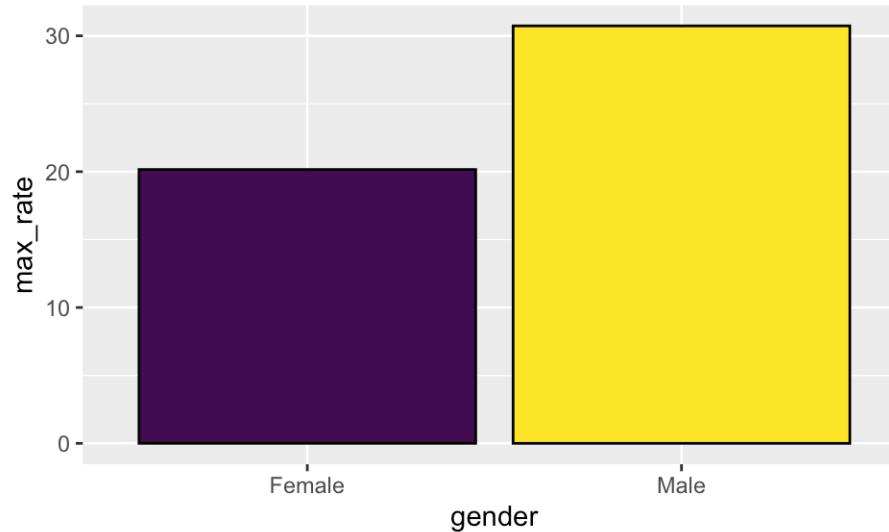
```
er_bar
```



Tip - color outside of aes()

Can be used to add an outline around column/bar plots.

```
  +  
  geom_col(color = "black")
```



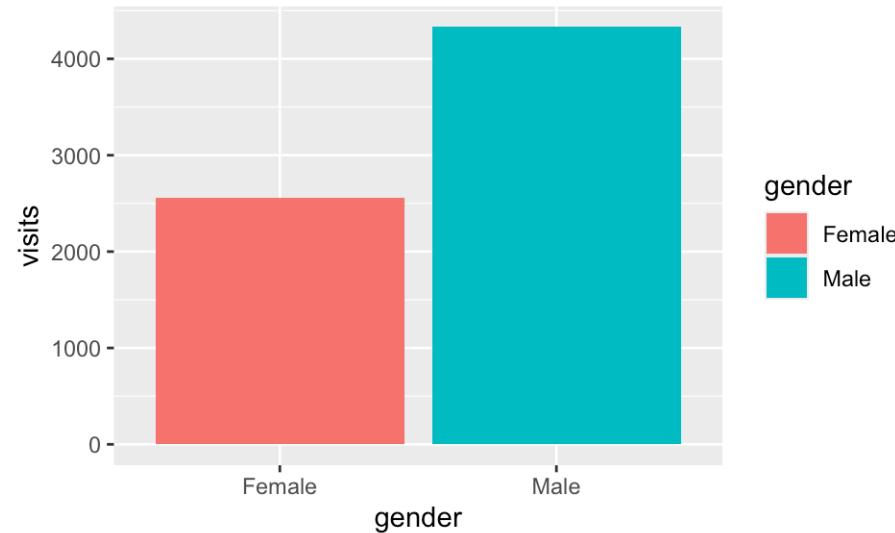
Tip - col vs bar

`geom_bar(x =)` can only use one aes mapping `geom_col(x = , y =)` can have two

Tip - Check what you plot

⚠ May not be plotting what you think you are! ⚠

```
ggplot(er_visits_gender, aes(x = gender,  
                               y = visits,  
                               fill = gender)) +  
  geom_col()
```



What did we plot? Always good to check it is correct!

```
head(er_visits_gender, n = 3)
```

```
# A tibble: 3 × 7
  county   rate lower95cl upper95cl visits   year gender
  <chr>    <dbl>     <dbl>     <dbl>    <dbl> <dbl> <chr>
1 Adams     7.60      4.38     11.7      17  2011 Female
2 Adams     NA         NA        NA       NA  2012 Female
3 Adams     6.22      3.37     9.93      14  2013 Female
```

```
er_visits_gender %>% group_by(gender) %>%
  summarize(sum = sum(visits, na.rm=T))
```

```
# A tibble: 2 × 2
  gender     sum
  <chr>    <dbl>
1 Female    2556
2 Male      4331
```

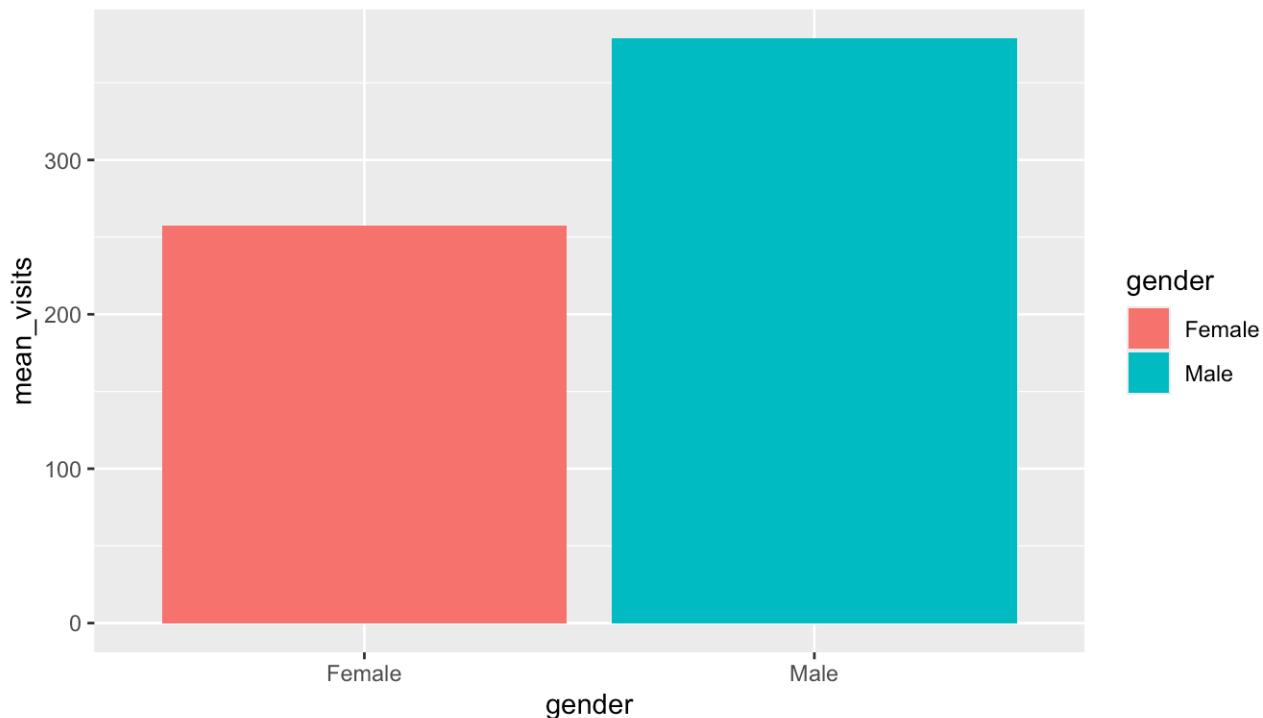
Try that again

```
er_visits_gender %>% group_by(gender, county) %>%  
  summarize(mean_visits = mean(visits, na.rm=T))
```

```
# A tibble: 20 × 3  
# Groups: gender [2]  
  gender county    mean_visits  
  <chr>  <chr>        <dbl>  
1 Female Adams      15.8  
2 Female Arapahoe   14.4  
3 Female Cheyenne    0  
4 Female Denver     14.4  
5 Female El Paso    15.3  
6 Female Jefferson  14.1  
7 Female Larimer    13.5  
8 Female Pueblo     12.7  
9 Female Statewide   142.  
10 Female Weld       15  
11 Male   Adams     18.9  
12 Male   Arapahoe   17.3  
13 Male   Cheyenne    0  
14 Male   Denver     22.5  
15 Male   El Paso    23.1  
16 Male   Jefferson  16.3  
17 Male   Larimer    20.7  
18 Male   Pueblo     17.1  
19 Male   Statewide   225.  
20 Male   Weld       17.5
```

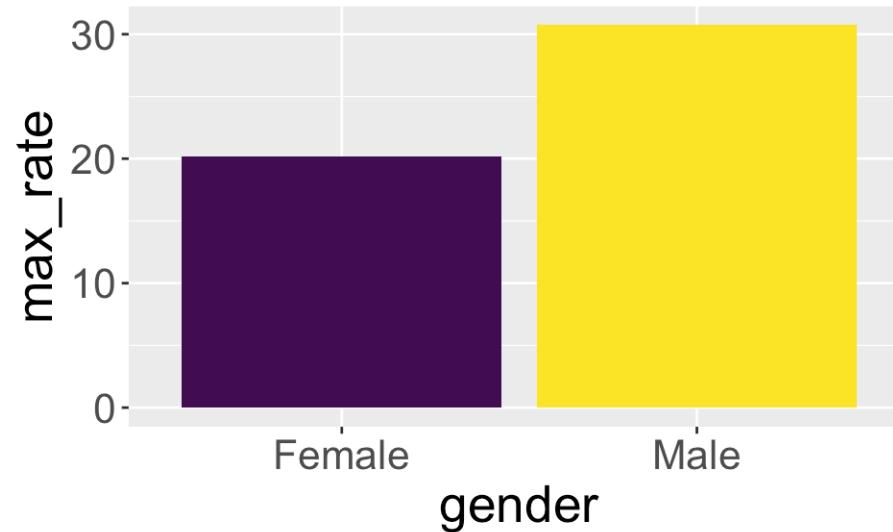
Try that again

```
er_visits_gender %>% group_by(gender, county) %>%  
  summarize(mean_visits = mean(visits, na.rm=T)) %>%  
  
ggplot(aes(x = gender,  
            y = mean_visits,  
            fill = gender)) +  
  geom_col()
```



Tip - make sure labels aren't too small

```
er_bar +  
  theme(text = element_text(size = 20))
```



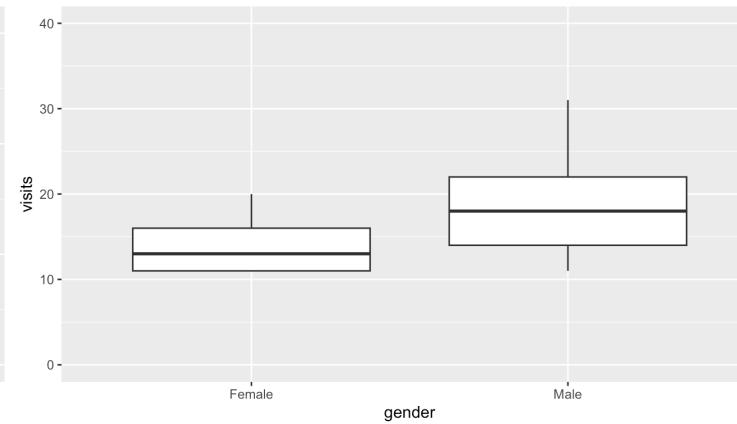
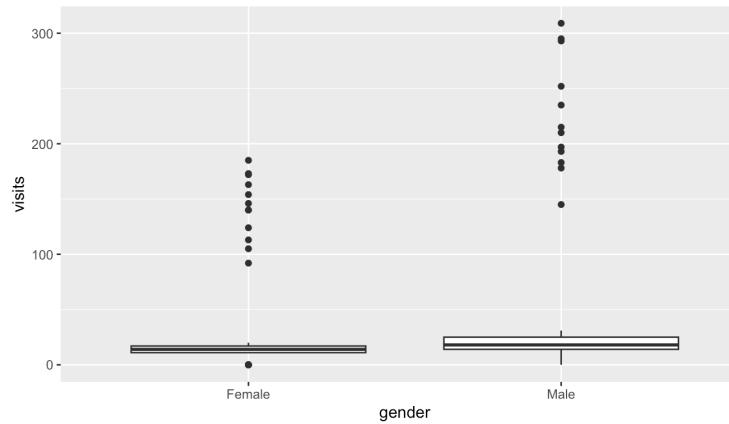
Tip- if you need you can remove outliers

Set `outlier.shape = NA` to get ride of outliers. Be careful about if you really should remove these!

However, it can be helpful if your plot is getting stretched to accommodate plotting an outlier. You can always say in the figure legend what you removed.

```
er_no_out1 <- ggplot(er_visits_gender, aes(y = visits, x = gender)) +  
  geom_boxplot()
```

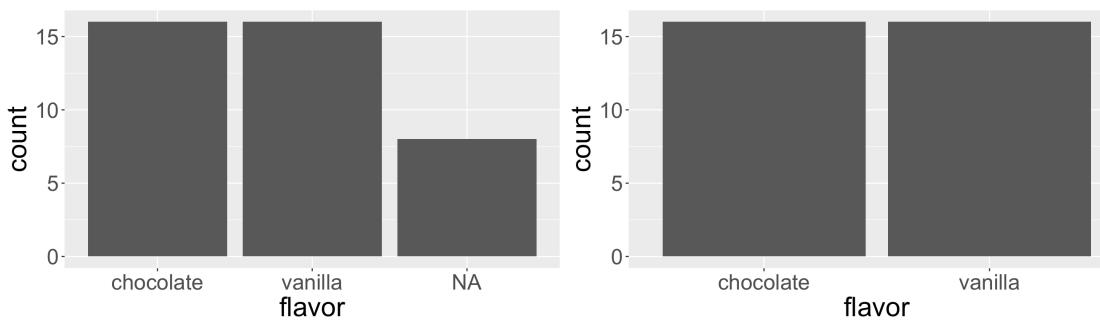
```
er_no_out2 <- ggplot(er_visits_gender, aes(y = visits, x = gender)) +  
  geom_boxplot(outlier.shape = NA) +  
  ylim(0, 40)
```



NA Values

- if it is a numeric value it will just get dropped from the graph and you will see a warning
- it is categorical you will see it on the graph and will need to filter to remove the NA category

```
icecream <- tibble(flavor =  
  rep(c("chocolate", "vanilla", NA, "chocolate", "vanilla"), 8))  
  
icecream1 <- ggplot(icecream, aes(x = flavor)) + geom_bar() +  
  theme(text=element_text(size=24))  
  
icecream2 <- icecream %>% drop_na(flavor) %>%  
  ggplot( aes(x = flavor)) + geom_bar() +  
  theme(text=element_text(size=24))
```



Sometimes we have many lines and it is hard to see what is happening{.codesmall}

```
er_visits_9 <- er_co_county %>%  
  filter(county %in% c("Denver", "Weld", "Pueblo", "Jackson",  
    "San Juan", "Mesa", "Jefferson", "Larimer", "Statewide")) /
```

Adding a facet can help make it easier to see what is happening

Sometimes we have too many lines and can get difficult to see what is happening, facets can help!

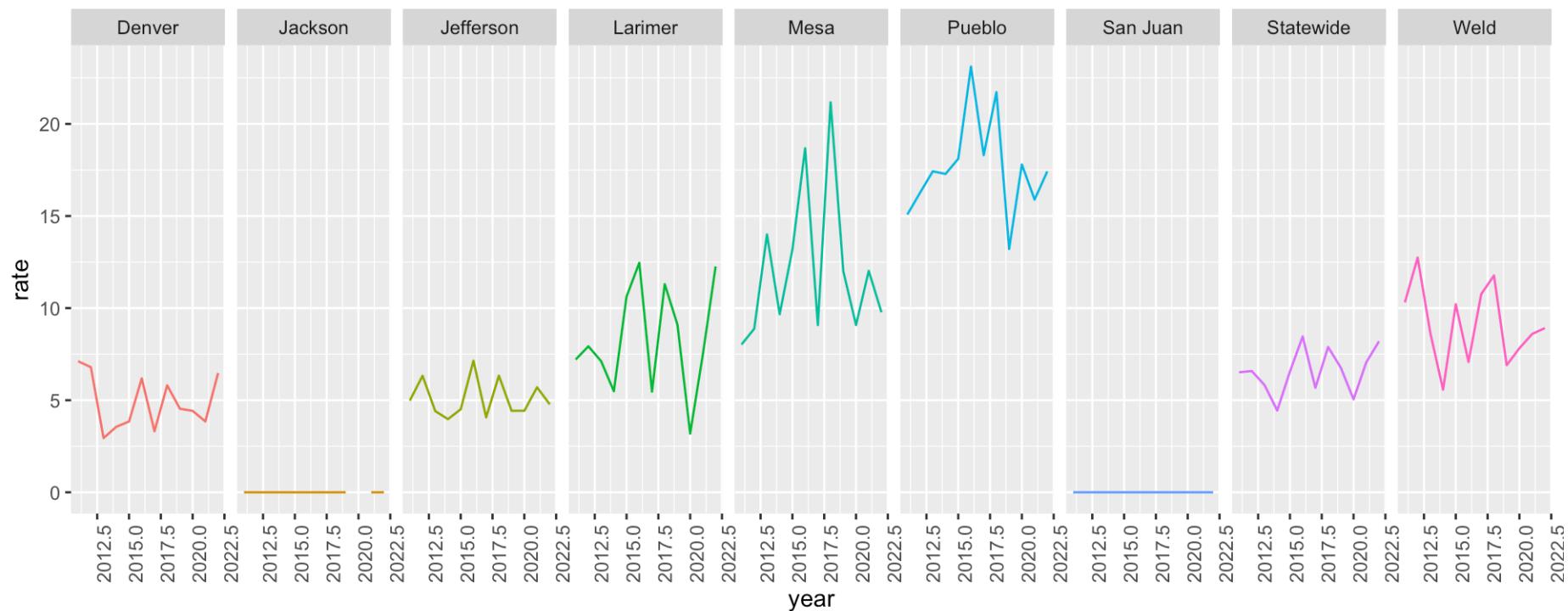
Two options: `facet_grid()`- creates a grid shape `facet_wrap()` -more flexible

Need to specify how you are faceting with the ~ sign.

```
lots_of_lines +  
facet_grid(~ county) +  
theme(legend.position = "bottom")
```

Adding a facet can help make it easier to see what is happening

```
lots_of_lines +  
facet_grid(~ county) +  
theme(legend.position = "none") +  
theme(axis.text.x = element_text(angle = 90))
```

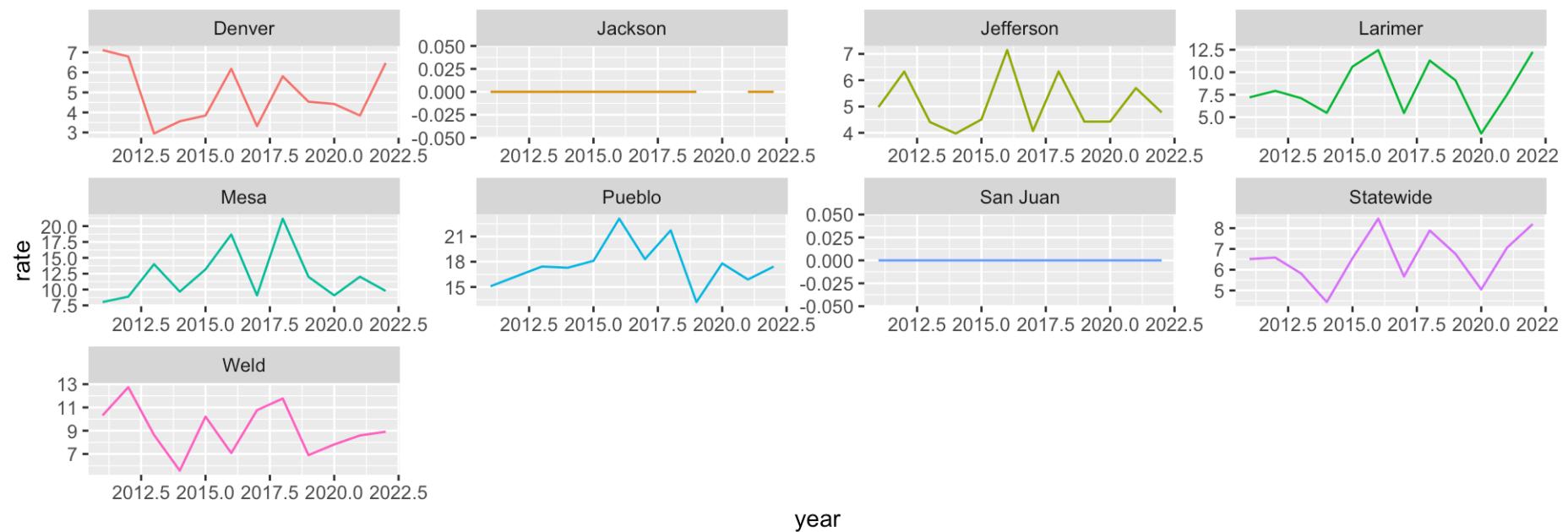


facet_wrap()

- more flexible - arguments `ncol` and `nrow` can specify layout
- can have different scales for axes using `scales = "free"`

```
rp_fac_plot <- lots_of_lines +  
  facet_wrap(~ county, ncol = 4, scales = "free") +  
  theme(legend.position = "none")
```

```
rp_fac_plot
```

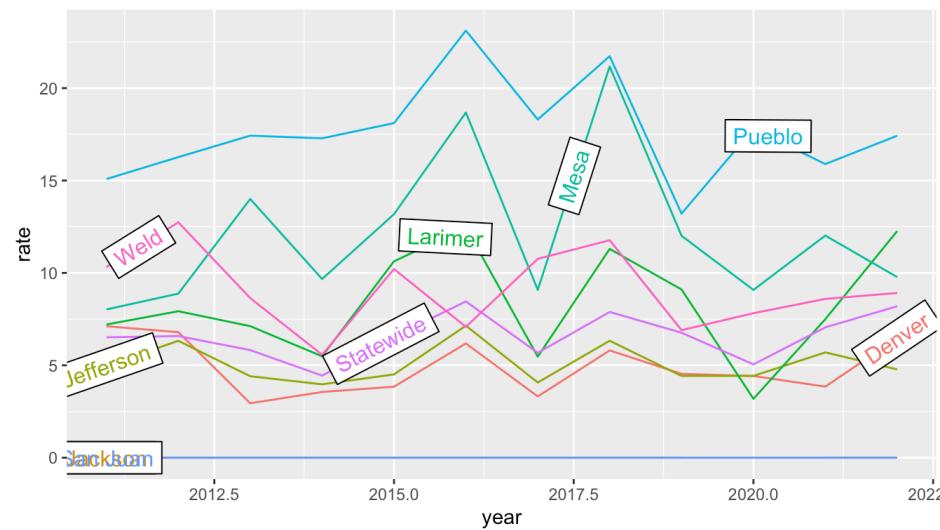


Extensions

directlabels package

Great for adding labels directly onto plots <https://www.opencasestudies.org/ocs-bp-co2-emissions/>

```
#install.packages("directlabels")
library(directlabels)
direct.label(lots_of_lines, method = list("angled.boxes"))
```

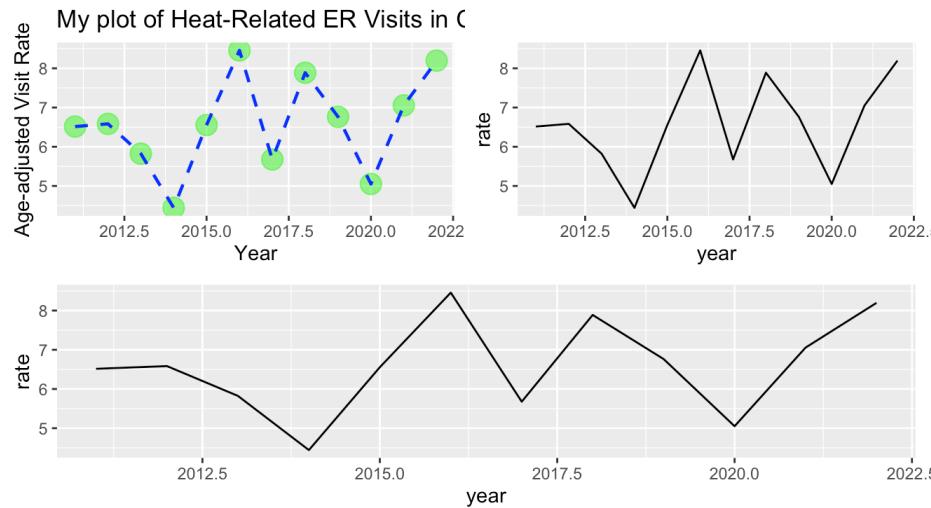


patchwork package

Great for combining plots together

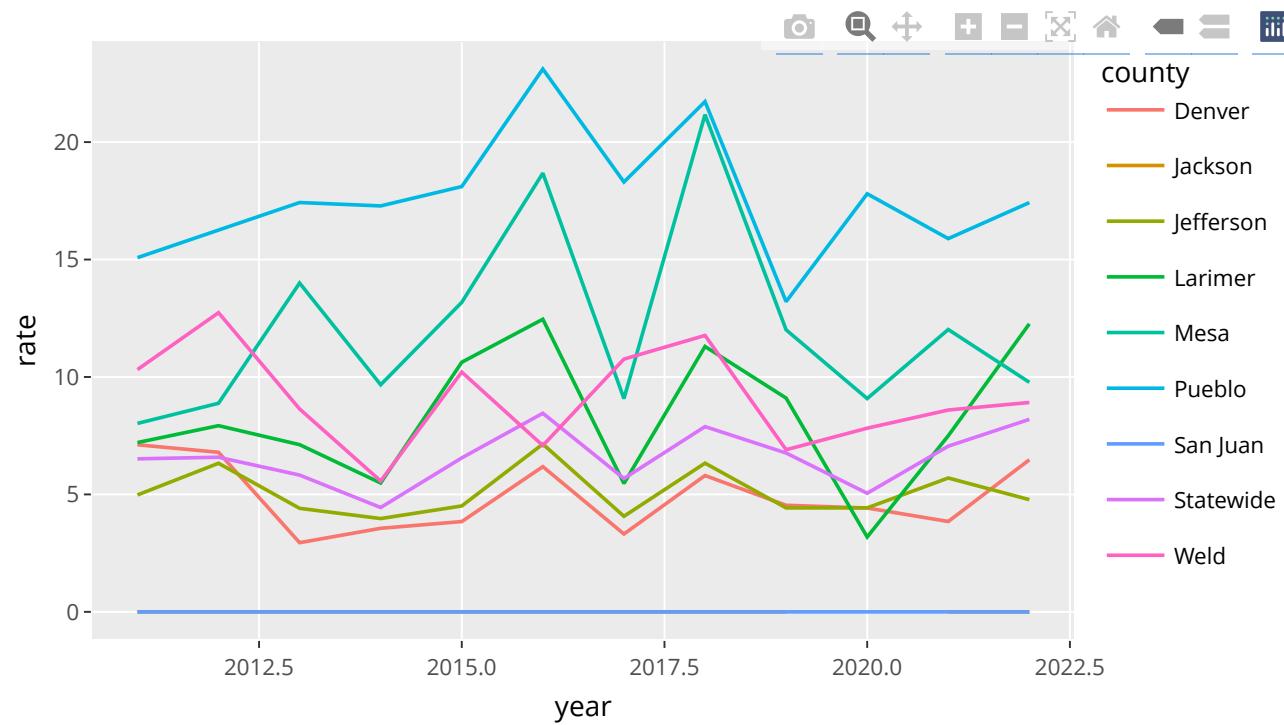
Also check out the [patchwork package](#)

```
#install.packages("patchwork")
library(patchwork)
(plt1 + plt2)/plt2
```



plotly

```
#install.packages("plotly")
library("plotly") # creates interactive plots!
ggplotly(lots_of_lines)
```



Also check out the [ggiraph package](#)

Saving plots

Saving a ggplot to file

A few options:

- RStudio > Plots > Export > Save as image / Save as PDF
- RStudio > Plots > Zoom > [right mouse click on the plot] > Save image as
- In the code

```
ggsave(filename = "saved_plot.png", # will save in working directory  
       plot = rp_fac_plot,  
       width = 6, height = 3.5)           # by default in inches
```

Summary

- The `theme()` function helps you specify aspects about your plot
 - move or remove a legend with `theme(legend.position = "none")`
 - change font aspects of individual text elements `theme(plot.title = element_text(size = 20))`
 - center a title: `theme(plot.title = element_text(hjust = 0.5))`
- sometimes you need to add a `group` element to `aes()` if your plot looks strange
- make sure you are plotting what you think you are by checking the numbers!
- `facet_grid(~ variable)` and `facet_wrap(~variable)` can be helpful to quickly split up your plot
 - `facet_wrap()` allows for a `scales = "free"` argument so that you can have a different axis scale for different plots
- use `fill` to fill in boxplots

Good practices for plots

Check out this [guide](#) for more information!

Lab 2

[Class Website](#)
[Lab](#)



Image by [Gerd Altmann](#) from [Pixabay](#)