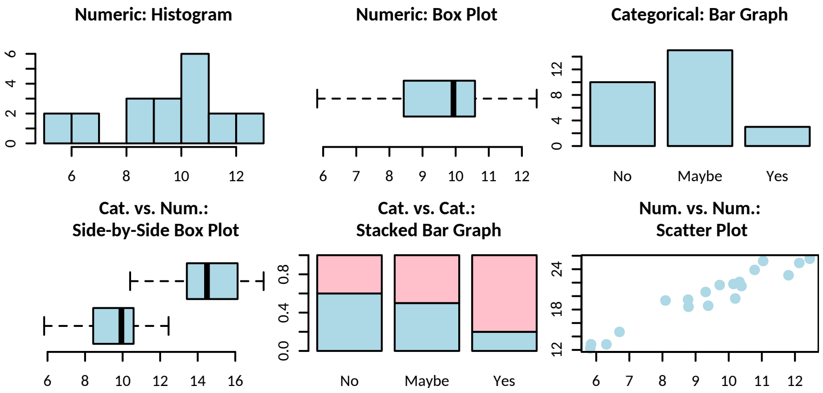
TRUMAN STATE UNIVERSITY

Department of
STATISTICS

Principles of `ggplot`

DISTINCT BY DESIGN

The type of variable(s) determines appropriate graphical representations.



Useful Base-R Graphing Commands

- `plot`
  - When in doubt, give this one a try.
  - Will make boxplots, scatter plots, bar graphs, from raw data.
  - Many more complex objects have `plot` methods.
- `hist` for histograms.
- `boxplot` for boxplots.
- `barplot` for bar graphs.

The Grammar of Graphics

The “Grammar of Graphics,” as implemented by `ggplot`, attempts to systematize data visualizations.

Most every data visualization can be created by specifying…

layer

- Data data frame
- Mapping var → aesthetic element
- “Geom” type of graph
- “Stat” stats needed by geom.

- Scales Var → page
- Coordinate Frame

- Faceting, and

by group

- Theme Misc.

The `ggplot` package implements this grammar in R.

Although the basics aren’t hard, it helps to really understand how `ggplot` “thinks” in order to use its full potential.

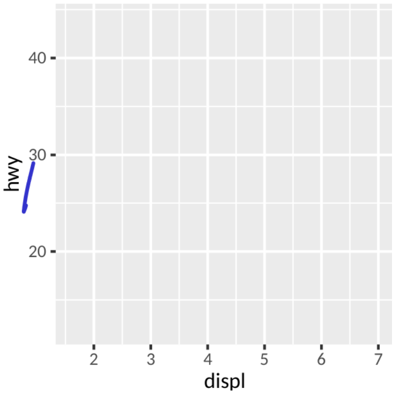
I find most help I need by tab-completing to find these keywords.

? geom_ <TAB>

? scale_

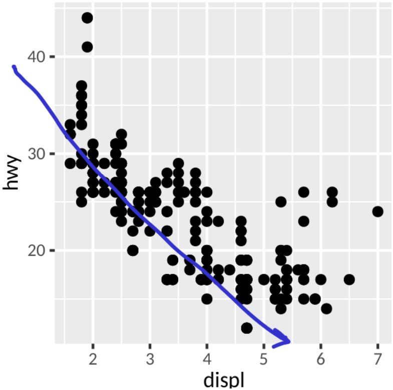
Data and mapping can be specified in the ggplot() command. They are inherited in subsequent layers.

```r
library(tidyverse)
ggplot(data=mpg,
  mapping=aes(x=displ, y=hwy))
```



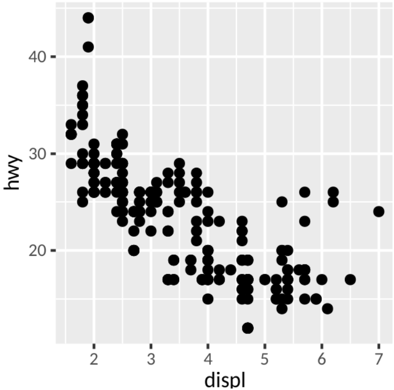
A **layer** is specified by giving a `geom` and a `stat`, but the command isn't usually called directly.

```r
library(tidyverse)
ggplot(data=mpg,
  mapping=aes(x=displ, y=hwy)) +
layer(geom="point",
  stat="identity",
  position="identity")
```



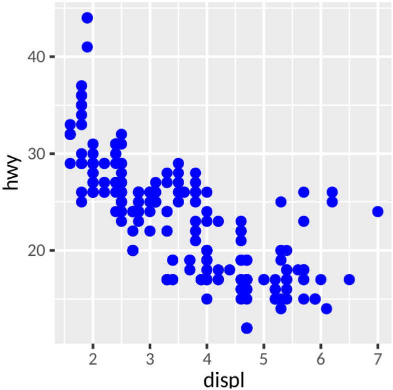
The `geom_` commands are shortcuts. Each has a default `stat`.

```
library(tidyverse)
ggplot(data=mpg,
  mapping=aes(x=displ, y=hwy)) +
geom_point()
```



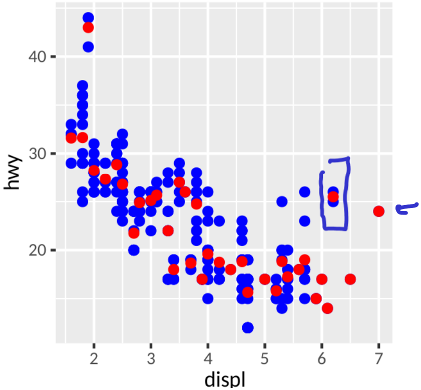
Aesthetics specified outside the `aes` command are constant.

```r
library(tidyverse)
ggplot(data=mpg,
  mapping=aes(x=displ, y=hwy)) +
geom_point(color="blue")
```



The stat calculates the statistics needed by the geom from the raw data. You can override the default.

```
library(tidyverse)
ggplot(data=mpg,
  mapping=aes(x=displ, y=hwy)) +
geom_point(color="blue") +
geom_point(color="red",
  stat="summary", fun=mean)
```

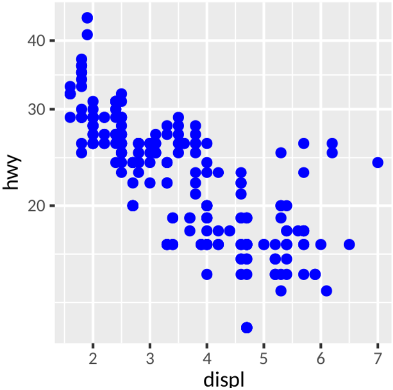


Scales determine how the mapped variables are transformed into a representation "on the page."

```
library(tidyverse)
ggplot(data=mpg,
  mapping=aes(x=displ, y=hwy)) +
geom_point(color="blue") +
scale_y_log10()
```

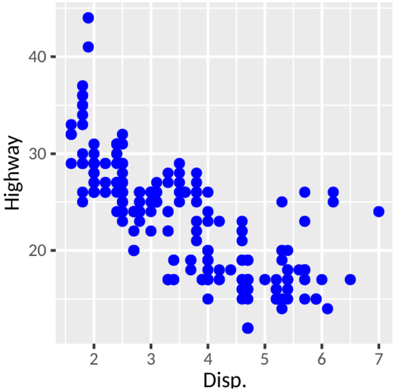
$\log(30)$

$\log_{10}(20)$



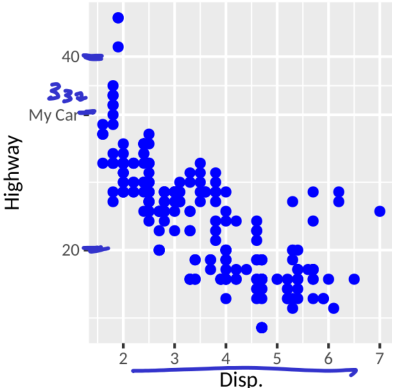
There is a generic `scale` function, but many specific `scale_` versions. Here we set the scale `name`.

```r
library(tidyverse)
ggplot(data=mpg,
  mapping=aes(x=displ, y=hwy)) +
geom_point(color="blue") +
scale_y_continuous(name="Highway") +
scale_x_continuous(name="Disp.")
```



You can set a scale's `breaks` and `labels`.

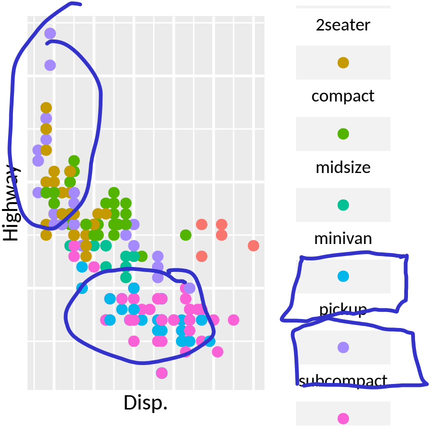
```
library(tidyverse)
ggplot(data=mpg,
  mapping=aes(x=displ, y=hwy)) +
geom_point(color="blue") +
scale_y_continuous(name="Highway",
  breaks=c(20, 33, 40),
  labels=c("20", "My Car", "40")) +
scale_x_continuous(name="Disp.")
```



Axes and legends are scale **guides**. You can adjust them with the `guide` option.

Class

```
library(tidyverse)
ggplot(data=mpg,
  mapping=aes(x=displ, y=hwy)) +
geom_point(aes(color=class)) +
scale_y_continuous(
  name="Highway",
  guide=NULL) +
scale_x_continuous(
  name="Disp.",
  guide=NULL) +
scale_color_discrete(
  name="Class",
  guide=guide_legend(
    label.position="bottom"
  ))
```

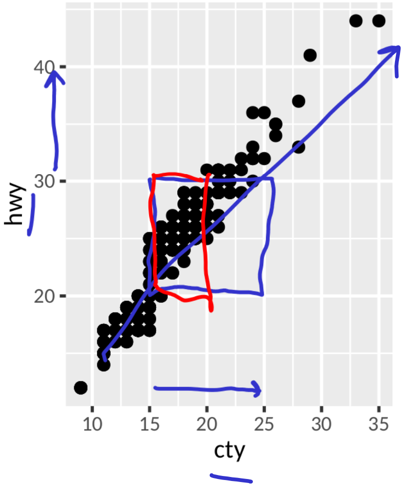


Legend

Coordinates specify how x and y are mapped to the plane of the page. `coord_fixed` controlls the aspect ratio.

```
ggplot(mpg,
  mapping=aes(x=cty, y=hwy)) +
geom_point() +
coord_fixed(ratio=1)
```

? coord_ <TAB>

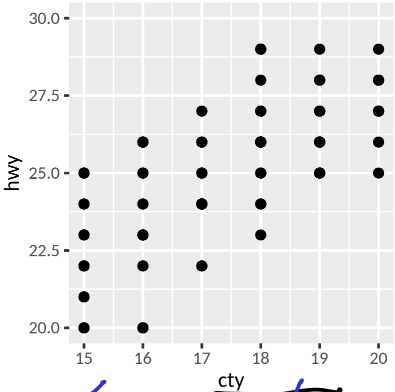


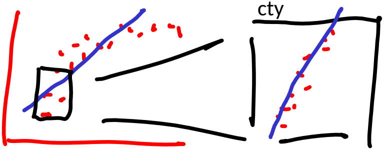
The coordinates_ commands also give the right way to zoom in on part of the picture.

```
ggplot(mpg,
  mapping=aes(x=cty, y=hwy)) +
geom_point() +
coord_cartesian(
  xlim=c(15,20),
  ylim=c(20,30))
```

just zoom

specify limits in scale — through out data pts not in those limits



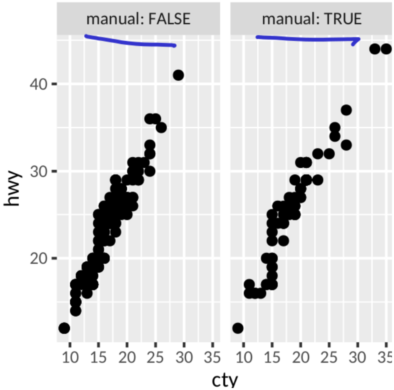


Facets split the data frame, and create multiple data sets.

T/F

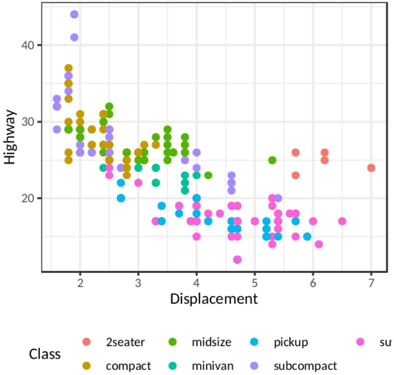
```
mpg %>% mutate(
  manual=grepl("manual", trans)) %>%
ggplot(~~mpg~~,
  mapping=aes(x=cty, y=hwy)) +
geom_point() +
facet_wrap(
  facets=vars(manual),
  labeller=label_both)
```

facet_grid



Themes control other aspects of the graph's visual display not related to layers or aethetics.

```
library(tidyverse)
ggplot(data=mpg,
  mapping=aes(x=displ, y=hwy)) +
geom_point(aes(color=class)) +
scale_y_continuous(
  name="Highway") +
scale_x_continuous(
  name="Displacement") +
scale_color_discrete(
  name="Class") +
theme_bw() +
theme(legend.position="bottom")
```



?theme

Going Further with `ggplot`

“Helper commands” simplify some tasks, but can make the zoo of `ggplot` commands seem more confusing.

- `geom_` and `stat_` commands replace the `layer` command.
- Specific label commands can be used instead of `scale`: `xlab`, `ylab`, `ggtitle`.
- “Cross-cutting” helpers set attritubes of multiple scales at once: `labs`, `guides`, `lims`. labs( )

scale_x_continuous(name = , guide =)

Scale_y_continuous(name = , guide =)

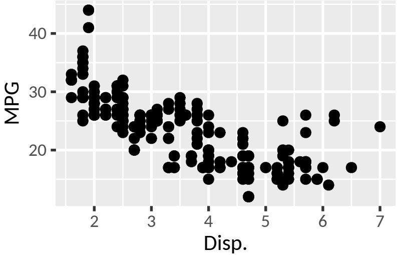
scale_color_discrete(name = , guide =)

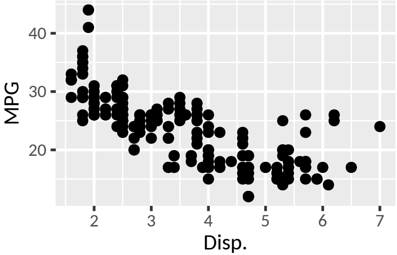
guides(x= ___ , y = ___ , color= ___)

An example of “cross-cutting” helper functions. The following code is equivalent.

```r
ggplot(mpg,
  mapping=aes(x=displ, y=hwy)) +
geom_point() +
scale_x_continuous(name="Disp.") +
scale_y_continuous(name="MPG")
```

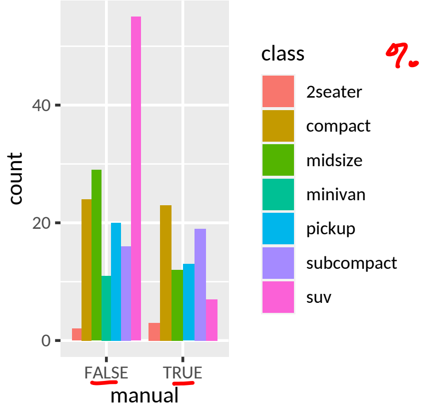
```r
# These helpers shorten the code.
ggplot(mpg,
  mapping=aes(x=displ, y=hwy)) +
geom_point() +
labs(x="Disp.", y="MPG")
```





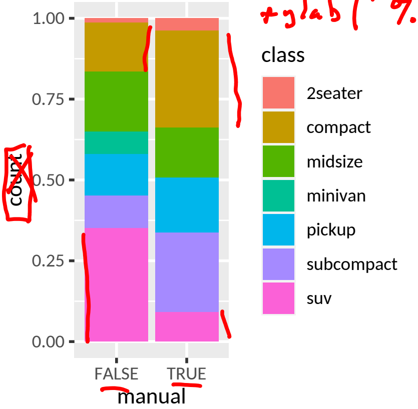
The position option controls how overlapping elements are handled. Especially important for bar graphs.

```r
mpg %>% mutate(
  manual=grepl("manual", trans)) %>%
ggplot(aes(x=manual, fill=class)) +
geom_bar(position="dodge")
```



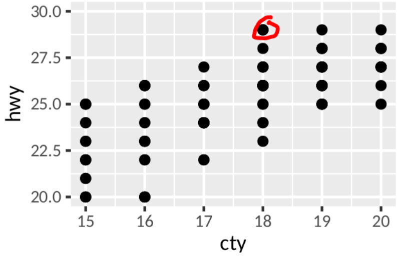
```r
mpg %>% mutate(
  manual=grepl("manual", trans)) %>%
ggplot(aes(x=manual, fill=class)) +
geom_bar(position="fill")
```

+ylab("%")

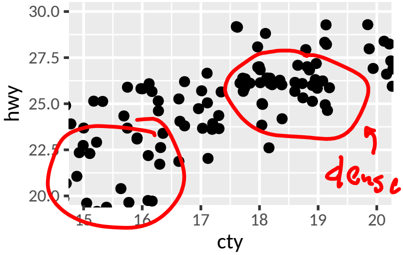


position="jitter" adds a bit of randomness to ponits that would otherwise overlap, for good or ill.

```
ggplot(mpg,
  mapping=aes(x=cty, y=hwy)) +
geom_point() +
coord_cartesian(
  xlim=c(15,20), ylim=c(20,30))
```

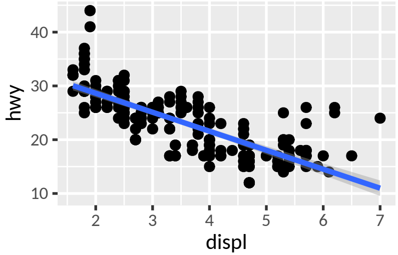


```
ggplot(mpg,
  mapping=aes(x=cty, y=hwy)) +
geom_point(position="jitter") +
coord_cartesian(
  xlim=c(15,20), ylim=c(20,30))
```

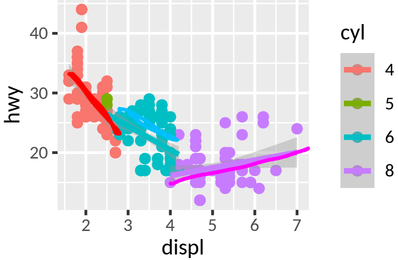


Setting an aesthetic to a factor variable defines a group in the data. This isn't always what we want.

```
# We want to fit a single line.
mpg %>%
mutate(cyl=factor(cyl)) %>%
ggplot(aes(x=displ, y=hwy)) +
geom_point() +
geom_smooth(method="lm")
```

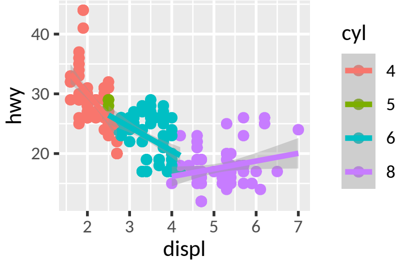


```
mpg %>%
mutate(cyl=factor(cyl)) %>%
ggplot(aes(x=displ, y=hwy,
  color=cyl)) +
geom_point() +
geom_smooth(method="lm")
```



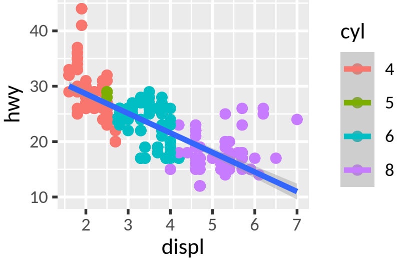
Specifying the “group” option will override the default grouping. Also defines groups without making a legend.

```r
# We don't want three lines!
mpg %>%
mutate(cyl=factor(cyl)) %>%
ggplot(aes(x=displ, y=hwy,
  color=cyl)) +
geom_point() +
geom_smooth(method="lm")
```



```r
mpg %>%
mutate(cyl=factor(cyl)) %>%
ggplot(aes(x=displ, y=hwy,
  color=cyl)) +
geom_point() +
geom_smooth(aes(group=1),
  method="lm")
```

group = "elephant"

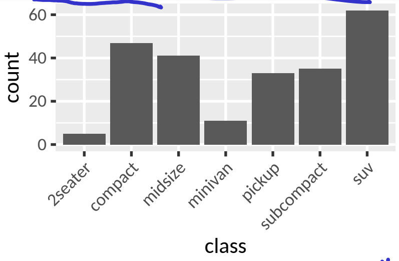


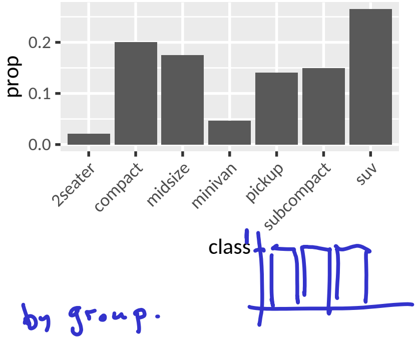
Statistics returned by the `stat` can be accessed directly, which is sometimes useful.

y= ..prop..

```
# Look at ?geom_bar to find the stat.
ggplot(data=mpg, aes(x=class)) +
geom_bar() +
theme(axis.text.x=
  element_text(angle=45, hjust=1))
```

```
ggplot(data=mpg, aes(x=class)) +
geom_bar(aes(y=after_stat(prop),
  group=1)) +
theme(axis.text.x=
  element_text(angle=45, hjust=1))
```

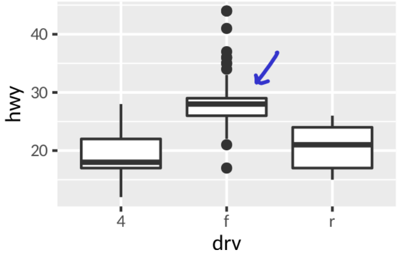




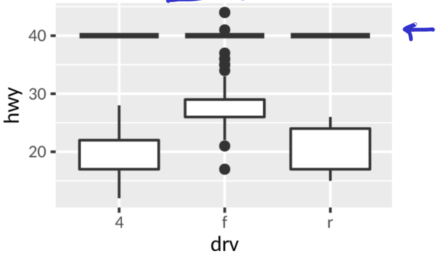
stat_count → count
→ prop % by group.

Statistics returned by the `stat` can be modified directly, which is usually dangerous.

```r
ggplot(mpg, aes(x=drv, y=hwy)) +
geom_boxplot()
```



```r
ggplot(mpg, aes(x=drv, y=hwy)) +
geom_boxplot(middle=40)
```



stat_boxplot → "middle"

→ Q,

→ outliers

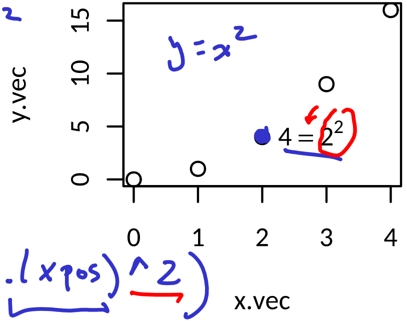
Did you know you can do mathematical typesetting in R?

- Check out `?plotmath`.
- It suggests using `expression` in your commands.
- I think bquote is better because you can access value of R variables in expressions with the `.()` construction.

```
x.vec <- 0:4; y.vec <- x.vec^2
xpos <- 2; ypos <- 4
plot(x=x.vec, y=y.vec)
text(x=xpos, y=ypos, pos=4,
  labels=bquote(.(ypos) == .(xp
```

Right of pt

M^2



expression(   )

bquote(   )

labels = bquote(.(ypos) == .(xpos)^2)