Part I

Syllabus and Software

Chapter 0

Introduction to R, Rstudio, and ggplot

Contents

0.1 R building blocks

R as calculator In the following sections, look at the commands and output and understand how the command was interpretted to give the output.

```
#### Calculator
# Arithmetic
2 * 10
```

```
## [1] 20
```

```
1 + 2
```

```
## [1] 3
```

```
# Order of operations is preserved
1 + 5 * 10
```

```
## [1] 51
```

```
(1 + 5) * 10
```

```
## [1] 60
```

```
# Exponents use the ^ symbol
2^5
```

```
## [1] 32
```

```
9^(1/2)
```

```
## [1] 3
```

Vectors A vector is a set of numbers like a column of a spreadsheet. Below these sets of numbers are not technically "vectors", since they are not in a row/column structure, though they are ordered and can be referred to by index.

```
#### Vectors
# Create a vector with the c (short for combine) function
c(1, 4, 6, 7)
```

```
## [1] 1 4 6 7
```

```
c(1:5, 10)
```

```
## [1]  1  2  3  4  5 10
```

```
# or use a function
# (seq is short for sequence)
seq(1, 10, by = 2)
```

```
## [1] 1 3 5 7 9
```

```
seq(0, 50, length = 11)
```

```
##  [1]  0  5 10 15 20 25 30 35 40 45 50
```

```
seq(1, 50, length = 11)
```

```
##  [1]  1.0  5.9 10.8 15.7 20.6 25.5 30.4 35.3 40.2 45.1 50.0
```

```
1:10 # short for seq(1, 10, by = 1), or just
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
seq(1, 10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
5:1
```

```
## [1] 5 4 3 2 1
```

```
# non-integer sequences
#   (Note: the [1] at the beginning of lines indicates
#          the index of the first value in that row)
seq(0, 100*pi, by = pi)
```

```
##   [1]   0.000000   3.141593   6.283185   9.424778  12.566371
##   [6]  15.707963  18.849556  21.991149  25.132741  28.274334
##  [11]  31.415927  34.557519  37.699112  40.840704  43.982297
```

```
##  [16]  47.123890  50.265482  53.407075  56.548668  59.690260
##  [21]  62.831853  65.973446  69.115038  72.256631  75.398224
##  [26]  78.539816  81.681409  84.823002  87.964594  91.106187
##  [31]  94.247780  97.389372 100.530965 103.672558 106.814150
##  [36] 109.955743 113.097336 116.238928 119.380521 122.522113
##  [41] 125.663706 128.805299 131.946891 135.088484 138.230077
##  [46] 141.371669 144.513262 147.654855 150.796447 153.938040
##  [51] 157.079633 160.221225 163.362818 166.504411 169.646003
##  [56] 172.787596 175.929189 179.070781 182.212374 185.353967
##  [61] 188.495559 191.637152 194.778745 197.920337 201.061930
##  [66] 204.203522 207.345115 210.486708 213.628300 216.769893
##  [71] 219.911486 223.053078 226.194671 229.336264 232.477856
##  [76] 235.619449 238.761042 241.902634 245.044227 248.185820
##  [81] 251.327412 254.469005 257.610598 260.752190 263.893783
##  [86] 267.035376 270.176968 273.318561 276.460154 279.601746
##  [91] 282.743339 285.884931 289.026524 292.168117 295.309709
##  [96] 298.451302 301.592895 304.734487 307.876080 311.017673
## [101] 314.159265
```

Assign variables Assigning objects to varibles.

```
#### Assign variables
# Assign a vector to a variable with <-
a <- 1:5
a
```

```
## [1] 1 2 3 4 5
```

```
b <- seq(15, 3, length = 5)
b
```

```
## [1] 15 12  9  6  3
```

```
c <- a * b
c
```

```
## [1] 15 24 27 24 15
```

Basic functions There are lots of functions available in the base package.

Type `?base` and click on `Index` at the bottom of the help page for a complete list of functions. Other functions to look at are in the `?stats` and `?datasets` packages.

```
#### Basic functions
# Lots of familiar functions work
a
```

```
## [1] 1 2 3 4 5
```

```
sum(a)
```

```
## [1] 15
prod(a)
## [1] 120
mean(a)
## [1] 3
sd(a)
## [1] 1.581139
var(a)
## [1] 2.5
min(a)
## [1] 1
median(a)
## [1] 3
max(a)
## [1] 5
range(a)
## [1] 1 5
```

Extracting subsets It will be an extremely important skill to understand the structure of your variables and pull out the information you need from them.

```
#### Extracting subsets
# Specify the indices you want in the square brackets []
a <- seq(0, 100, by = 10)
# blank = include all
a
##  [1]   0  10  20  30  40  50  60  70  80  90 100
a[]
##  [1]   0  10  20  30  40  50  60  70  80  90 100
# integer +=include, 0=include none, -=exclude
a[5]
## [1] 40
a[c(2, 4, 6, 8)]
## [1] 10 30 50 70
a[0]
## numeric(0)
a[-c(2, 4, 6, 8)]
## [1]   0  20  40  60  80  90 100
```

```
a[c(1, 1, 1, 6, 6, 9)]   # subsets can be bigger than the original set
## [1]  0  0  0 50 50 80
a[c(1,2)] <- c(333, 555) # update a subset
a
##  [1] 333 555  20  30  40  50  60  70  80  90 100
```

Boolean: True/False Subsets can be selected based on which elements meet specific conditions.

```
#### Boolean
a
##  [1] 333 555  20  30  40  50  60  70  80  90 100
(a > 50)         # TRUE/FALSE for each element
##  [1]  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
which(a > 50)    # which indicies are TRUE
## [1]  1  2  7  8  9 10 11
a[(a > 50)]
## [1] 333 555  60  70  80  90 100
!(a > 50)        # ! negates (flips) TRUE/FALSE values
##  [1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
a[!(a > 50)]
## [1] 20 30 40 50
```

Comparison functions All your favorite comparisons are available.

```
#### Comparison
# Here they are:  <  >  <=  >=  !=  ==  %in%
a
##  [1] 333 555  20  30  40  50  60  70  80  90 100
# equal to
a[(a == 50)]
## [1] 50
# equal to
a[(a == 55)]
## numeric(0)
# not equal to
a[(a != 50)]
##  [1] 333 555  20  30  40  60  70  80  90 100
```

```
# greater than
a[(a > 50)]
## [1] 333 555  60  70  80  90 100
# less than
a[(a < 50)]
## [1] 20 30 40
# less than or equal to
a[(a <= 50)]
## [1] 20 30 40 50
# which values on left are in the vector on right
(c(10, 14, 40, 60, 99) %in% a)
## [1] FALSE FALSE  TRUE  TRUE FALSE
```

Boolean operators compare TRUE/FALSE values and return TRUE/-FALSE values.

```
#### Boolean
# & and, | or, ! not
a
##  [1] 333 555  20  30  40  50  60  70  80  90 100
a[(a >= 50) & (a <= 90)]
## [1] 50 60 70 80 90
a[(a < 50) | (a > 100)]
## [1] 333 555  20  30  40
a[(a < 50) | !(a > 100)]
## [1]  20  30  40  50  60  70  80  90 100
a[(a >= 50) & !(a <= 90)]
## [1] 333 555 100
```

Missing values The value `NA` (not available) means the value is missing. Any calculation involving `NA` will return an `NA` by default.

```
#### Missing values
NA + 8
## [1] NA
3 * NA
## [1] NA
mean(c(1, 2, NA))
```

```
## [1] NA
# Many functions have an na.rm argument  (NA remove)
mean(c(NA, 1, 2), na.rm = TRUE)
## [1] 1.5
sum(c(NA, 1, 2))
## [1] NA
sum(c(NA, 1, 2), na.rm = TRUE)
## [1] 3
# Or you can remove them yourself
a <- c(NA, 1:5, NA)
a
## [1] NA  1  2  3  4  5 NA
is.na(a)      # which values are missing?
## [1]  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE
!is.na(a)     # which values are NOT missing?
## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
a[!is.na(a)]  # return those which are NOT missing
## [1] 1 2 3 4 5
a             # note, this did not change the variable a
## [1] NA  1  2  3  4  5 NA
# To save the results of removing the NAs,
#   assign to another variable or reassign to the original variable
# Warning: if you write over variable a then the original version is gone forever!
a <- a[!is.na(a)]
a
## [1] 1 2 3 4 5
```

Your turn!

■ ClickerQs — Ch 0, R building blocks, Subset ■

■ ClickerQs — Ch 0, R building blocks, T/F selection 1 ■

■ CLICKER Qs — Ch 0, R building blocks, T/F selection 2 ■

How'd you do?

Outstanding Understanding the operations and how to put them together, without skipping steps.

Good Understanding most of the small steps, missed a couple details.

Hang in there Understanding some of the concepts but all the symbols make my eyes spin.

Reading and writing in a new language takes work.

You'll get better as you practice, practice, practice[1].

Having a buddy to work with will help.

R command review This is a summary of the commands we've seen so far.

```
#### Review
<-
+ - * / ^
c()
seq() # by=, length=
sum(), prod(), mean(), sd(), var(),
min(), median(), max(), range()
a[]
(a > 1), ==, !=, >, <, >=, <=, %in%
&, |, !
NA, mean(a, na.rm = TRUE), !is.na()
```

0.2 Plotting with ggplot2

There are three primary strategies for plotting in R. The first is base graphics, using functions such as `plot()`, `points()`, and `par()`. A second is use of the package `lattice`, which creates very nice graphics quickly, though can be more difficult to create high-dimensional data displays. A third, and the one I will use throughout this course, is using package `ggplot2`, which is an implementation of the "Grammar of Graphics". While creating a very simple plot requires an

[1] What's your Carnegie Hall that you're working towards?

extra step or two, creating very complex displays are relatively straightforward as you become comfortable with the syntax.

This section is intended as an introduction to `ggplot()` and some of its capabilities (we will cover these plotting functions and others in detail in later chapters). As a basic introduction, it requires a `data.frame` object as input (a table where each row represents an observation or data point, and each column can have a different data type), and then you define plot layers that stack on top of each other, and each layer has visual/text elements that are mapped to aesthetics (colors, size, opacity). In this way, a simple set of commands can be combined to produce extremely informative displays.

In the example that follows, we consider a dataset `mpg` consisting of fuel economy data from 1999 and 2008 for 38 popular models of car.

```
#### Installing
# only needed once after installing or upgrading R
install.packages("ggplot2")
```

```
#### Library
# each time you start R
# load package ggplot2 for its functions and datasets
library(ggplot2)

# ggplot2 includes a dataset "mpg"

# ? gives help on a function or dataset
?mpg
```

Let's get a look at the dataset we're using.

```
#### mpg dataset
# head() lists the first several rows of a data.frame
head(mpg)
```

```
## # A tibble: 6 x 11
##   manufacturer model displ  year   cyl trans drv     cty   hwy fl
##   <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr>
## 1 audi         a4      1.8  1999     4 auto~ f        18    29 p
## 2 audi         a4      1.8  1999     4 manu~ f        21    29 p
## 3 audi         a4      2    2008     4 manu~ f        20    31 p
## 4 audi         a4      2    2008     4 auto~ f        21    30 p
## 5 audi         a4      2.8  1999     6 auto~ f        16    26 p
## 6 audi         a4      2.8  1999     6 manu~ f        18    26 p
## # ... with 1 more variable: class <chr>
```

```
# str() gives the structure of the object
str(mpg)
```

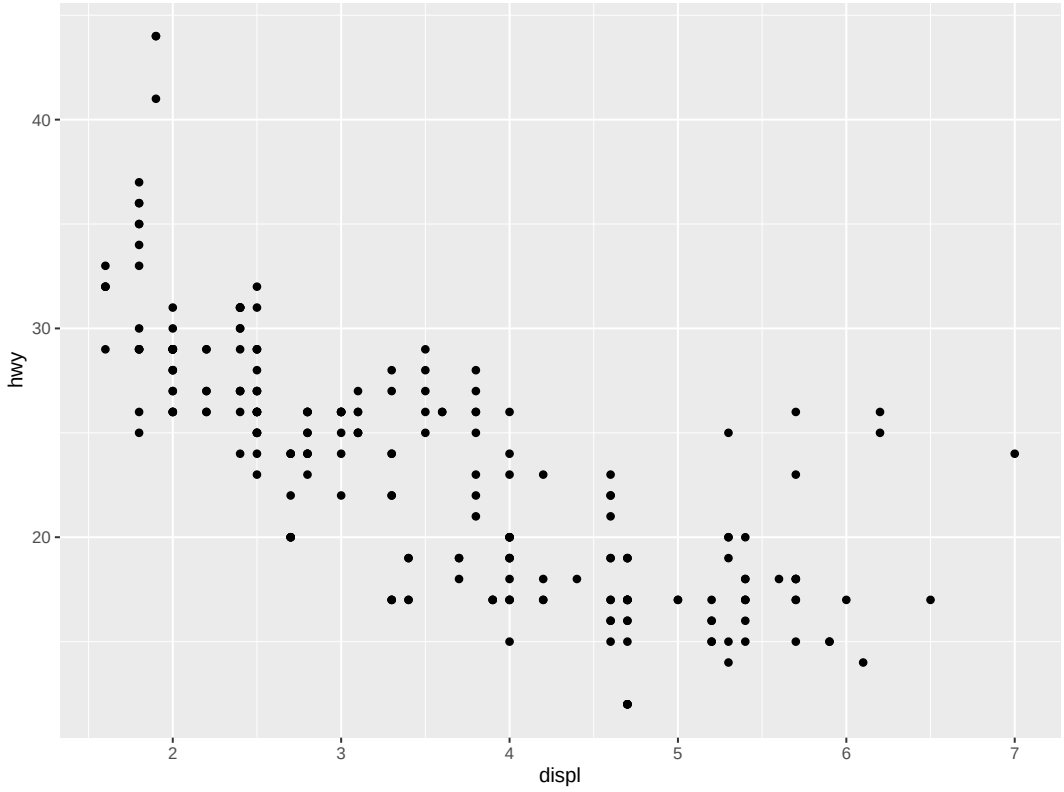
```
## Classes 'tbl_df', 'tbl' and 'data.frame': 234 obs. of  11 variables:
##  $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
##  $ model       : chr  "a4" "a4" "a4" "a4" ...
##  $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
##  $ year        : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
##  $ cyl         : int  4 4 4 4 6 6 6 4 4 4 ...
##  $ trans       : chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
##  $ drv         : chr  "f" "f" "f" "f" ...
##  $ cty         : int  18 21 20 21 16 18 18 18 16 20 ...
##  $ hwy         : int  29 29 31 30 26 26 27 26 25 28 ...
##  $ fl          : chr  "p" "p" "p" "p" ...
##  $ class       : chr  "compact" "compact" "compact" "compact" ...
```

```
# summary() gives frequeny tables for categorical variables
#              and mean and five-number summaries for continuous variables
summary(mpg)
```

```
##  manufacturer          model               displ            year
##  Length:234         Length:234         Min.   :1.600   Min.   :1999
##  Class :character   Class :character   1st Qu.:2.400   1st Qu.:1999
##  Mode  :character   Mode  :character   Median :3.300   Median :2004
##                                        Mean   :3.472   Mean   :2004
##                                        3rd Qu.:4.600   3rd Qu.:2008
##                                        Max.   :7.000   Max.   :2008
##       cyl           trans               drv
##  Min.   :4.000   Length:234         Length:234
##  1st Qu.:4.000   Class :character   Class :character
##  Median :6.000   Mode  :character   Mode  :character
##  Mean   :5.889
##  3rd Qu.:8.000
##  Max.   :8.000
##       cty             hwy             fl
##  Min.   : 9.00   Min.   :12.00   Length:234
##  1st Qu.:14.00   1st Qu.:18.00   Class :character
##  Median :17.00   Median :24.00   Mode  :character
##  Mean   :16.86   Mean   :23.44
##  3rd Qu.:19.00   3rd Qu.:27.00
##  Max.   :35.00   Max.   :44.00
##     class
##  Length:234
##  Class :character
##  Mode  :character
##
##
##
```

```
#### ggplot_mpg_displ_hwy
# specify the dataset and variables
p <- ggplot(mpg, aes(x = displ, y = hwy))
p <- p + geom_point() # add a plot layer with points
```

```
print(p)
```



Geoms, aesthetics, and facets are three concepts we'll see in this section.

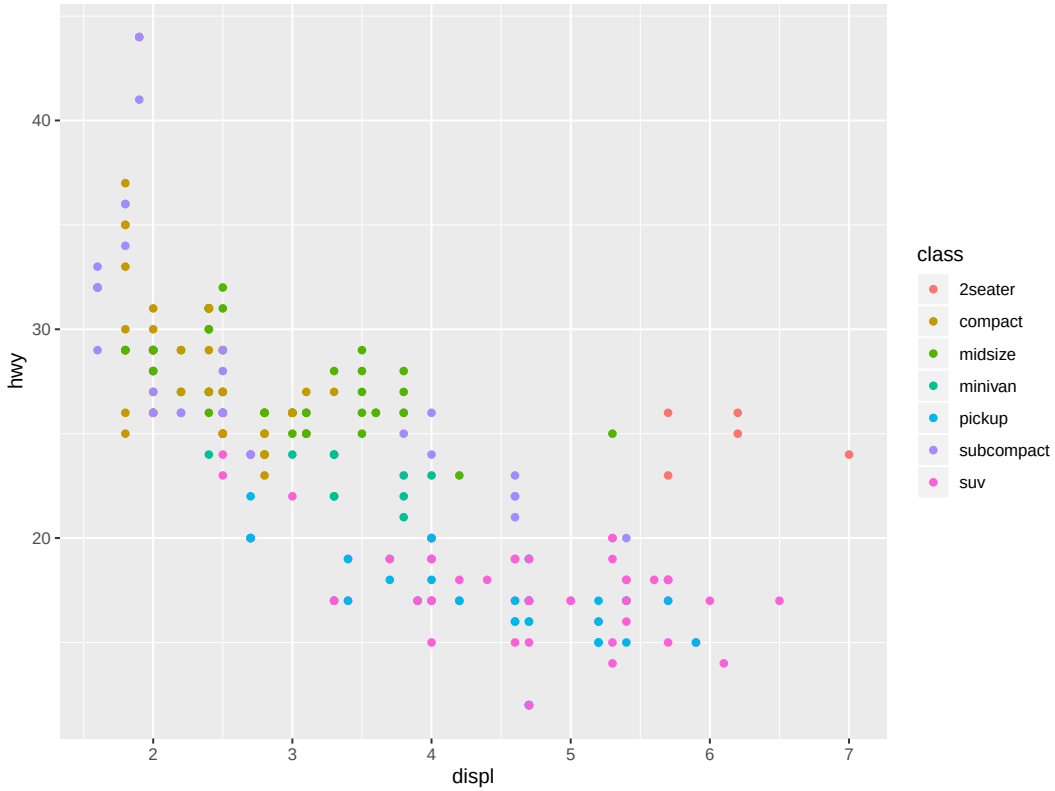
Geom: is the "type" of plot
Aesthetics: shape, colour, size, alpha
Faceting: "small multiples" displaying different subsets

Help is available. Try searching for examples, too.

- `had.co.nz/ggplot2`
- `had.co.nz/ggplot2/geom_point.html`

When certain aesthetics are defined, an appropriate legend is chosen and displayed automatically.

```
#### ggplot_mpg_displ_hwy_colour_class
p <- ggplot(mpg, aes(x = displ, y = hwy))
p <- p + geom_point(aes(colour = class))
print(p)
```



I encourage you to experiment with aesthetics!

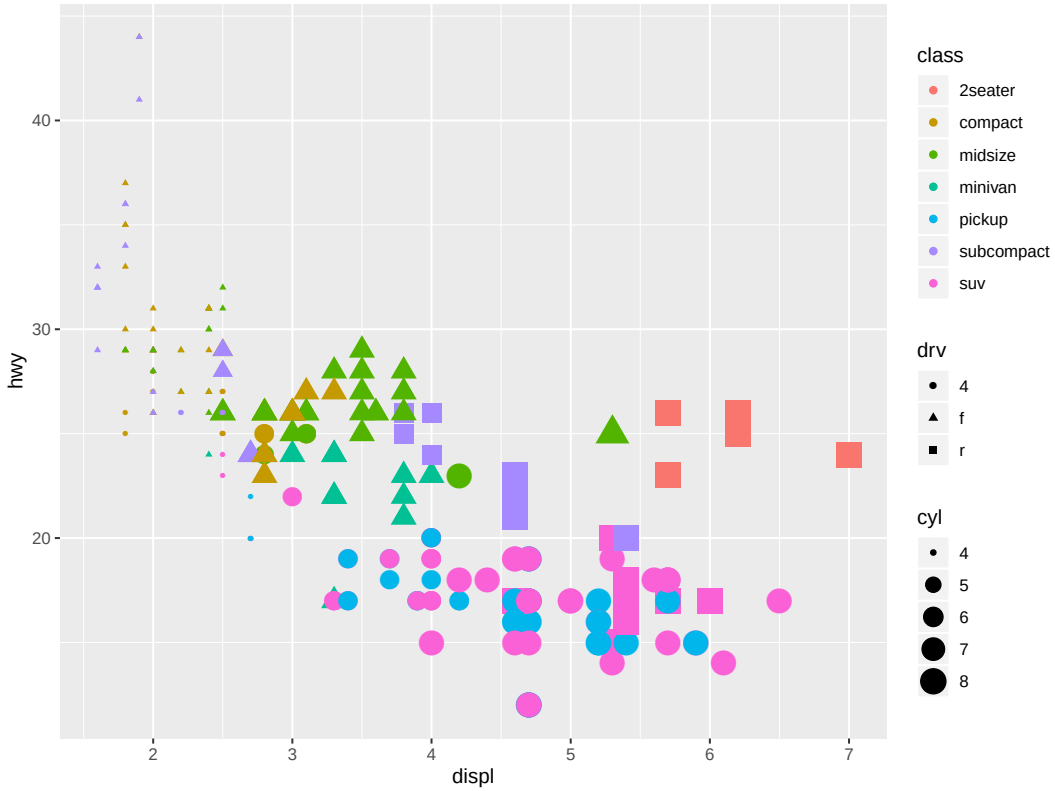
1. Assign variables to aesthetics colour, size, and shape.
2. What's the difference between discrete or continuous variables?
3. What happens when you combine multiple aesthetics?

The behavior of the aesthetics is predictable and customizable.

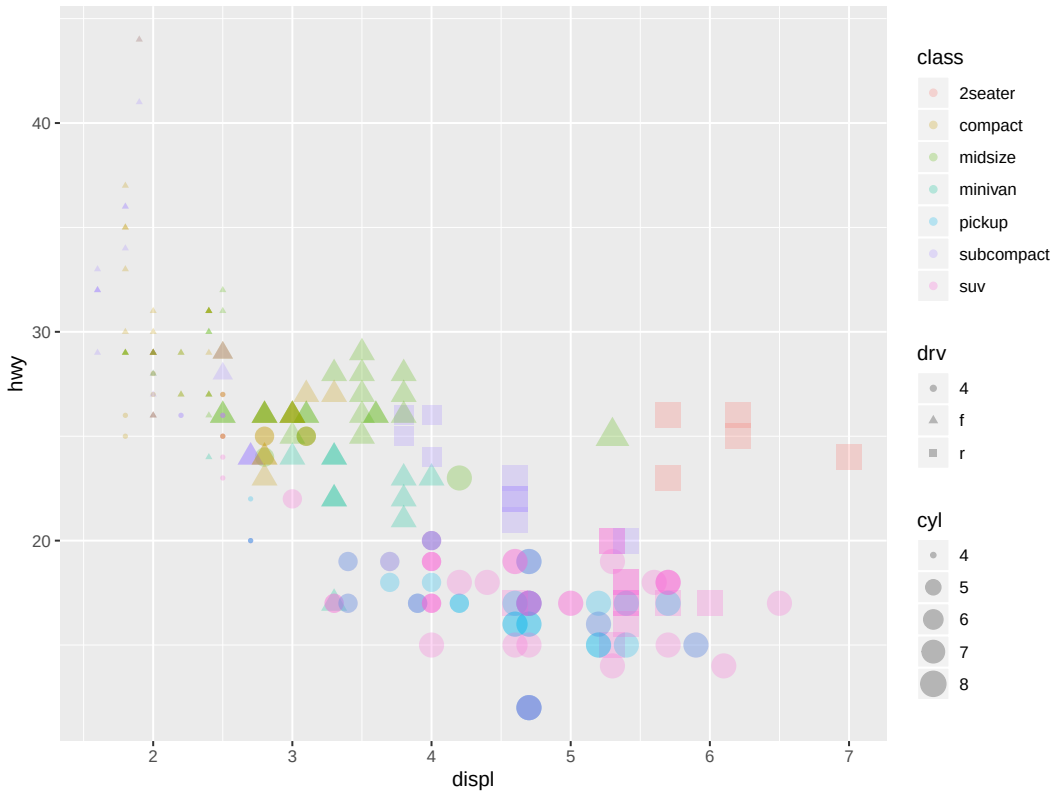
Aesthetic	Discrete	Continuous
colour	Rainbow of colors	Gradient from red to blue
size	Discrete size steps	Linear mapping between radius and value
shape	Different shape for each	Shouldn't work

Let's see a couple examples.

```
#### ggplot_mpg_displ_hwy_colour_class_size_cyl_shape_drv
p <- ggplot(mpg, aes(x = displ, y = hwy))
p <- p + geom_point(aes(colour = class, size = cyl, shape = drv))
print(p)
```




```
#### ggplot_mpg_displ_hwy_colour_class_size_cyl_shape_drv_alpha
p <- ggplot(mpg, aes(x = displ, y = hwy))
p <- p + geom_point(aes(colour = class, size = cyl, shape = drv)
                    , alpha = 1/4) # alpha is the opacity
print(p)
```



Faceting A small multiple[2] (sometimes called faceting, trellis chart, lattice chart, grid chart, or panel chart) is a series or grid of small similar graphics or charts, allowing them to be easily compared.

- Typically, small multiples will display different subsets of the data.
- Useful strategy for exploring conditional relationships, especially for large data.

Experiment with faceting of different types. What relationships would you like to see?

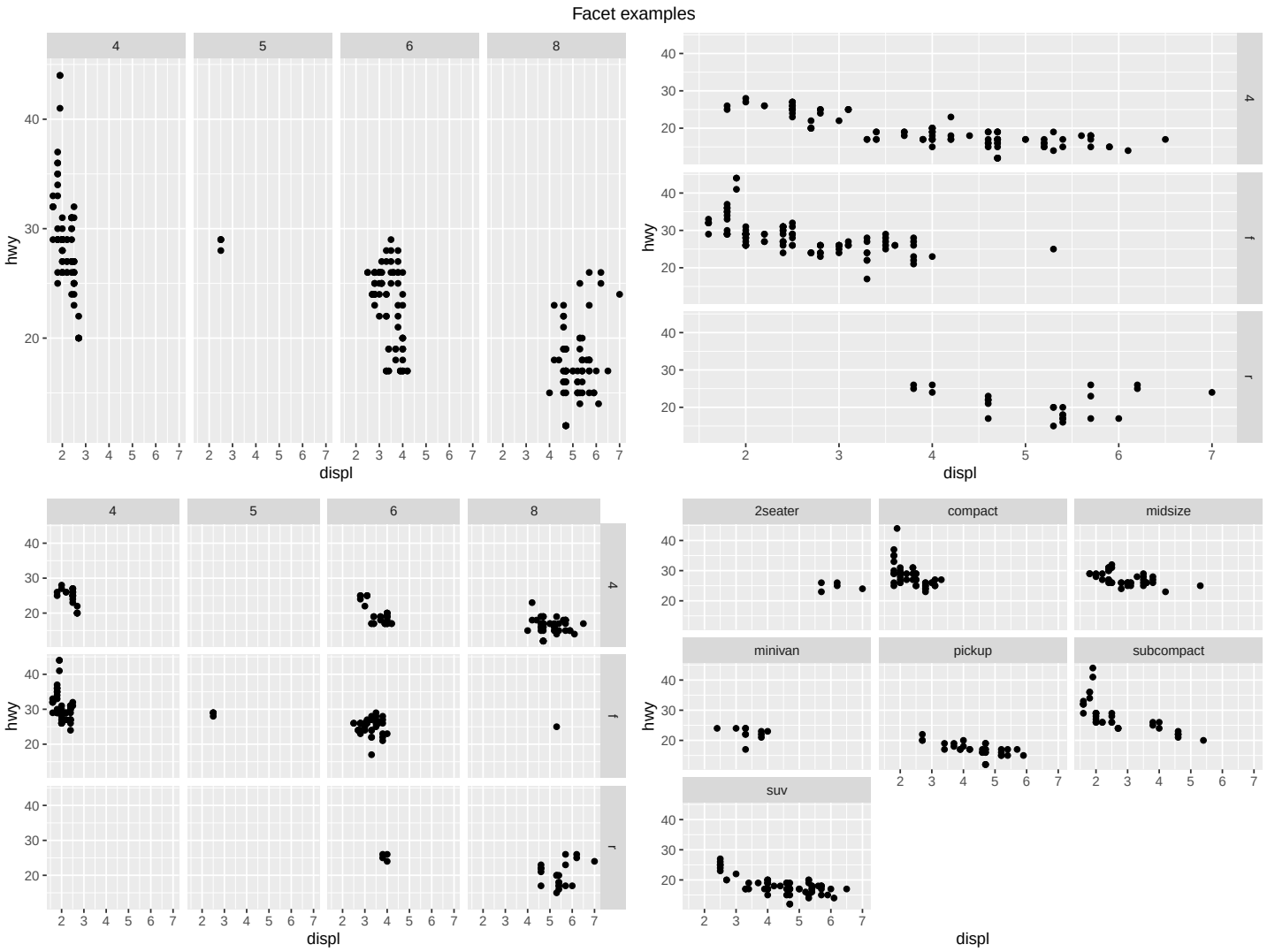
```
#### ggplot_mpg_displ_hwy_facet
# start by creating a basic scatterplot
p <- ggplot(mpg, aes(x = displ, y = hwy))
p <- p + geom_point()

## two methods
# facet_grid(rows ~ cols) for 2D grid, "." for no split.
# facet_wrap(~ var)       for 1D ribbon wrapped into 2D.

# examples of subsetting the scatterplot in facets
p1 <- p + facet_grid(. ~ cyl)     # columns are cyl categories
p2 <- p + facet_grid(drv ~ .)     # rows are drv categories
p3 <- p + facet_grid(drv ~ cyl)   # both
p4 <- p + facet_wrap(~ class)     # wrap plots by class category

# plot all four in one arrangement
library(gridExtra)
grid.arrange(grobs = list(p1, p2, p3, p4), ncol = 2, top="Facet examples")
```

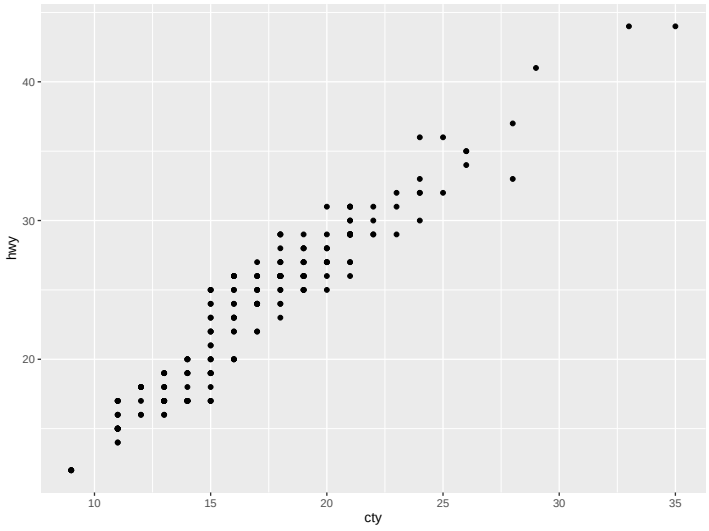
[2]According to Edward Tufte (Envisioning Information, p. 67): "At the heart of quantitative reasoning is a single question: Compared to what? Small multiple designs, multivariate and data bountiful, answer directly by visually enforcing comparisons of changes, of the differences among objects, of the scope of alternatives. For a wide range of problems in data presentation, small multiples are the best design solution."



0.2.1 Improving plots

How can this plot be improved?

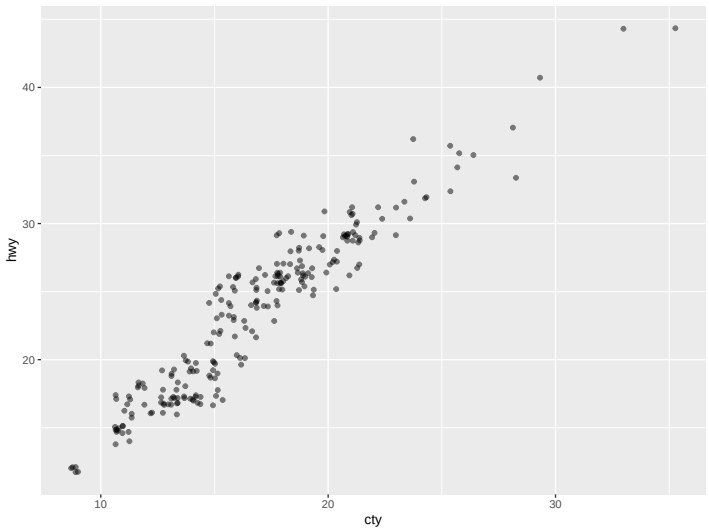
```
#### ggplot_mpg_cty_hwy
p <- ggplot(mpg, aes(x = cty, y = hwy))
p <- p + geom_point()
print(p)
```



Problem: points lie on top of each other, so it's impossible to tell how many observations each point represents.

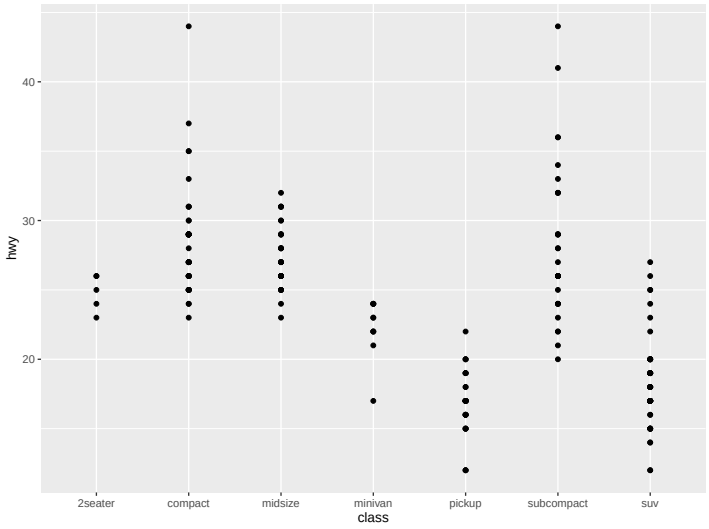
A solution: Jitter the points to reveal the individual points and reduce the opacity to 1/2 to indicate when points overlap.

```
#### ggplot_mpg_cty_hwy_jitter
p <- ggplot(mpg, aes(x = cty, y = hwy))
p <- p + geom_point(position = "jitter", alpha = 1/2)
print(p)
```



How can this plot be improved?

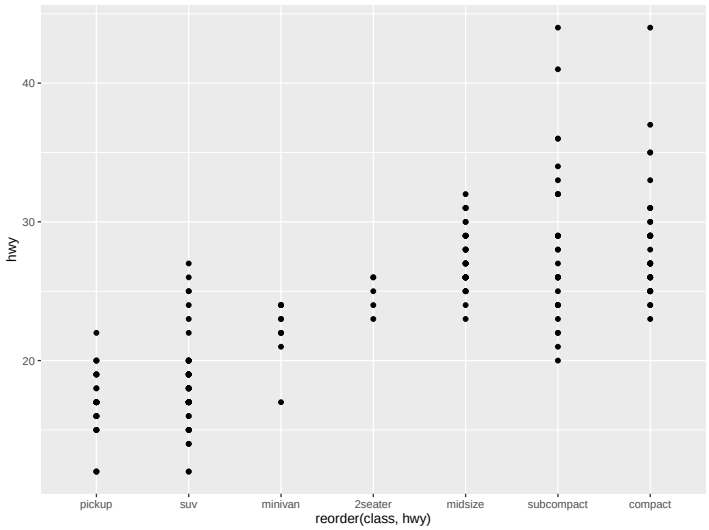
```
#### ggplot_mpg_class_hwy
p <- ggplot(mpg, aes(x = class, y = hwy))
p <- p + geom_point()
print(p)
```



Problem: The classes are in alphabetical order, which is somewhat arbitrary.

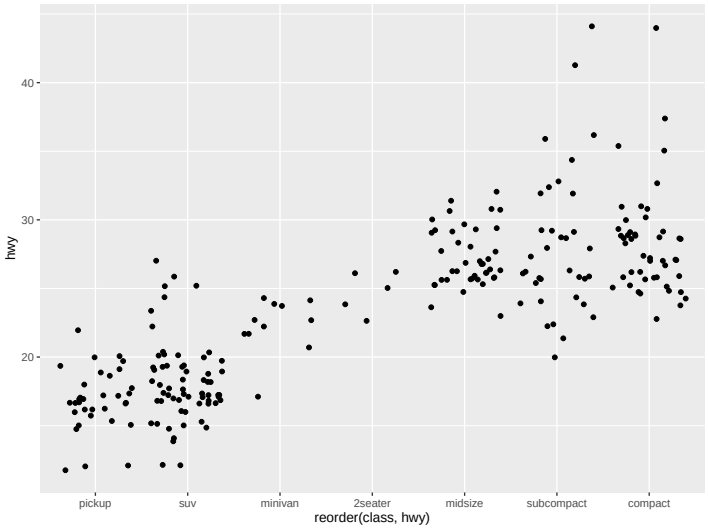
A solution: Reorder the class variable by the mean hwy for a meaningful ordering. Get help with `?reorder` to understand how this works.

```
#### ggplot_mpg_reorder_class_hwy
p <- ggplot(mpg, aes(x = reorder(class, hwy), y = hwy))
p <- p + geom_point()
print(p)
```



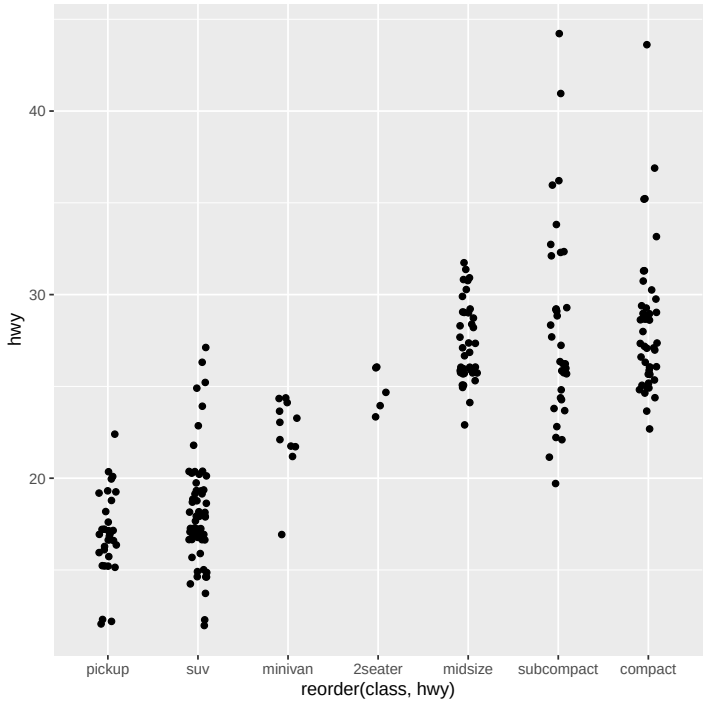
. . . add jitter

```
#### ggplot_mpg_reorder_class_hwy_jitter
p <- ggplot(mpg, aes(x = reorder(class, hwy), y = hwy))
p <- p + geom_point(position = "jitter")
print(p)
```



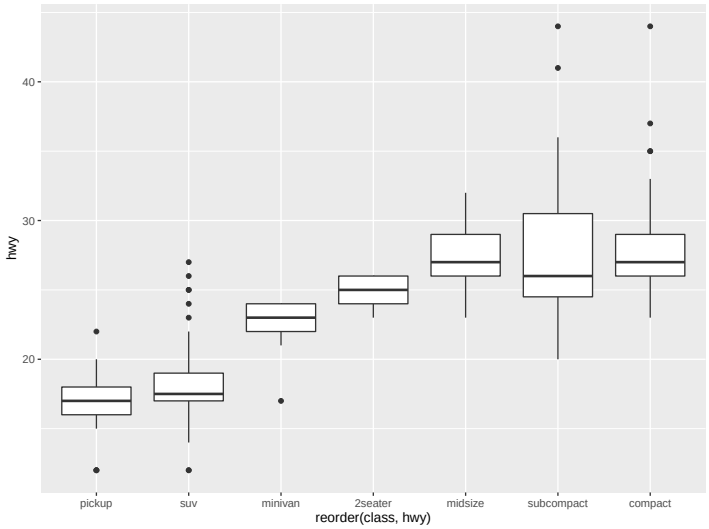
. . . a little less jitter

```
#### ggplot_mpg_reorder_class_hwy_jitter_less
p <- ggplot(mpg, aes(x = reorder(class, hwy), y = hwy))
p <- p + geom_jitter(position = position_jitter(width = 0.1))
print(p)
```



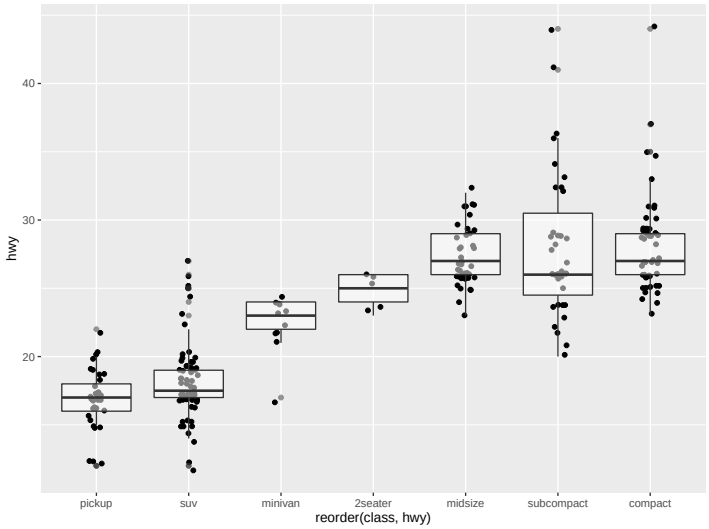
. . . or replace with boxplots

```
#### ggplot_mpg_reorder_class_hwy_boxplot
p <- ggplot(mpg, aes(x = reorder(class, hwy), y = hwy))
p <- p + geom_boxplot()
print(p)
```



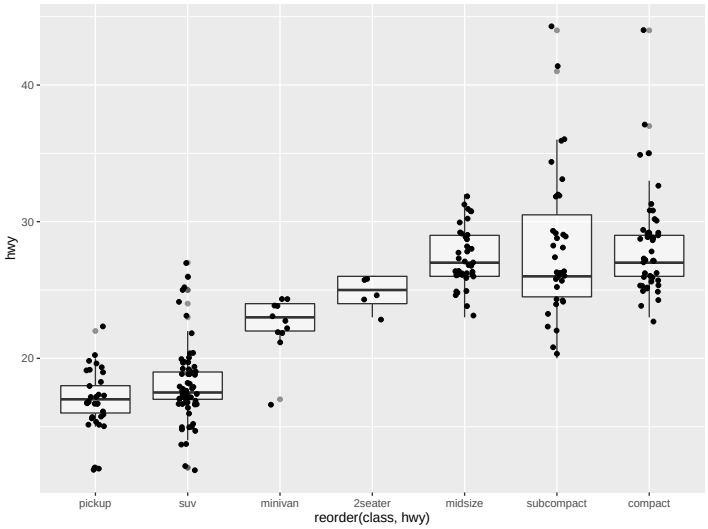
. . . or jitter those points with reduced-opacity boxplots on top

```
#### ggplot_mpg_reorder_class_hwy_jitter_boxplot
p <- ggplot(mpg, aes(x = reorder(class, hwy), y = hwy))
p <- p + geom_jitter(position = position_jitter(width = 0.1))
p <- p + geom_boxplot(alpha = 0.5)
print(p)
```



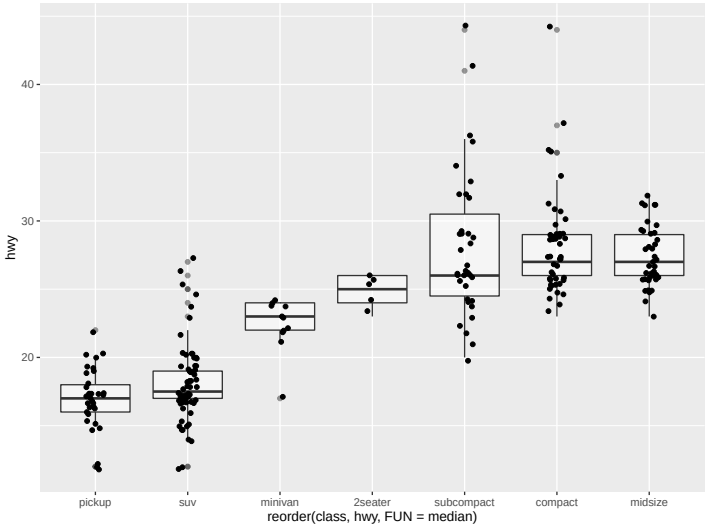
. . . even better with the boxplots with jittered points on top

```
#### ggplot_mpg_reorder_class_hwy_boxplot_jitter
p <- ggplot(mpg, aes(x = reorder(class, hwy), y = hwy))
p <- p + geom_boxplot(alpha = 0.5)
p <- p + geom_jitter(position = position_jitter(width = 0.1))
print(p)
```



. . . and can easily reorder by `median()` instead of `mean()` (mean is the default)

```
#### ggplot_mpg_reorder_class_hwy_boxplot_jitter_median
p <- ggplot(mpg, aes(x = reorder(class, hwy, FUN = median), y = hwy))
p <- p + geom_boxplot(alpha = 0.5)
p <- p + geom_jitter(position = position_jitter(width = 0.1))
print(p)
```



One-minute paper:

Muddy Any "muddy" points — anything that doesn't make sense yet?

Thumbs up Anything you really enjoyed or feel excited about?

0.3 Course Overview

See ADA2 Chapter 1 notes for a brief overview of all we'll cover in this semester of ADA1.

Part II

Summaries and displays, and one-, two-, and many-way tests of means

Part III

Nonparametric, categorical, and regression methods

Part IV

Additional topics