Day One: Cleaning and Visualization

Dillon Niederhut

December 9, 2015

1. Cleaning Data

introduction

there are two major steps to data cleaning, which we will call 'sanitizing' and 'tidying'

in sanitizing, our goal is to take each variable and force its values to be honest representations of its levels

in tidying, we are arranging our data structurally such that each row contains exactly one observation, and each column contains exactly one kind of data about that observation (this is sometimes expressed in SQL terms as "An attribute must tell something about the key, the whole key, and nothing but the key, so help me Codd")

exporting data from other software can do weird things to numbers and factors

it's usually better to DISABLE R's intuition about data types

unless you already know the data is clean and has no non-factor strings in it (i.e. you are the one who created it)

exporting data from other software can do weird things to numbers and factors

```
dirty <- read.csv('data/dirty.csv')
str(dirty)
```

```
## 'data.frame':    5 obs. of  5 variables:
##  $ Timestamp                   : Factor w/ 5 levels "7/25/2015 10:08:41",..: 1 2 3 4 5
##  $ How.tall.are.you.           : Factor w/ 5 levels "156","2.1","5’9",..: 5 4 3 2 1
##  $ What.department.are.you.in.: Factor w/ 5 levels "  geology","999",..: 4 2 1 5 3
##  $ Are.you.currently.enrolled.: Factor w/ 3 levels "999","No","Yes": 3 3 1 2 1
##  $ What.is.your.birth.order.  : Factor w/ 3 levels "1","2","9,000": 1 1 2 3 2
```

it's usually better to DISABLE R's intuition about data types

unless you already know the data is clean and has no non-factor strings in it (i.e. you are the one who created it)

```
dirty <- read.csv('data/dirty.csv',stringsAsFactors = FALSE)
str(dirty)
```

```
## 'data.frame':    5 obs. of  5 variables:
##  $ Timestamp                   : chr  "7/25/2015 10:08:41" "7/25/2015 10:10:56" "7/25/2015 10:11:20"
##  $ How.tall.are.you.           : chr  "very" "70" "5’9" "2.1" ...
```

```
##  $ What.department.are.you.in.: chr  "Geology  " "999" "  geology" "goelogy" ...
##  $ Are.you.currently.enrolled.: chr  "Yes" "Yes" "999" "No" ...
##  $ What.is.your.birth.order.  : chr  "1" "1" "2" "9,000" ...
```

let's start by removing the empty rows and columns

> note - R 3.2.2 and later does this automatically in `read.table` via `blank.lines.skip` and `skipNul`

```r
dim(dirty)
```

```
## [1] 5 5
```

```r
Filter(function(x)!all(is.na(x)), dirty)
```

```
##             Timestamp How.tall.are.you. What.department.are.you.in.
## 1 7/25/2015 10:08:41              very                   Geology  
## 2 7/25/2015 10:10:56                70                         999
## 3 7/25/2015 10:11:20               5'9                     geology
## 4 7/25/2015 10:11:25               2.1                     goelogy
## 5 7/25/2015 10:11:29               156                      anthro
##   Are.you.currently.enrolled. What.is.your.birth.order.
## 1                         Yes                         1
## 2                         Yes                         1
## 3                         999                         2
## 4                          No                     9,000
## 5                         999                         2
```

```r
dim(dirty)
```

```
## [1] 5 5
```

you can replace variable names

and you should, if they are uninformative or long

```r
names(dirty)
```

```
## [1] "Timestamp"                   "How.tall.are.you."
## [3] "What.department.are.you.in." "Are.you.currently.enrolled."
## [5] "What.is.your.birth.order."
```

```r
names(dirty) <- c("time", "height", "dept", "enroll", "birth.order")
```

it's common for hand-coded data to have a signifier for subject-missingness

(to help differentiate it from your hand-coder forgetting to do something)

```
dirty$enroll
```

```
## [1] "Yes" "Yes" "999" "No"  "999"
```

you should replace all of these values in your dataframe with R's missingness signifier, `NA`

```
table(dirty$enroll)
```

```
##
## 999  No Yes
##   2   1   2
```

```
dirty$enroll[dirty$enroll=="999"] <- NA
table(dirty$enroll, useNA = "ifany")
```

```
##
##   No  Yes <NA>
##    1    2    2
```

that timestamp variable is not in a format R likes

base R doesn't handle time well, so we need to get rid of the time part of the timestamp

```
dirty$time
```

```
## [1] "7/25/2015 10:08:41" "7/25/2015 10:10:56" "7/25/2015 10:11:20"
## [4] "7/25/2015 10:11:25" "7/25/2015 10:11:29"
```

```
dirty$time <- sub(' [0-9]+:[0-9]+:[0-9]+','',dirty$time)
dirty$time
```

```
## [1] "7/25/2015" "7/25/2015" "7/25/2015" "7/25/2015" "7/25/2015"
```

let's fix some of those department spellings

first, let's make this all lowercase

```
dirty$dept
```

```
## [1] "Geology  " "999"        "  geology" "goelogy"   "anthro"
```

```
dirty$dept <- tolower(dirty$dept)
dirty$dept <- gsub(' ', '', dirty$dept)  # what did we just do?
dirty$dept[4] <- "geology"
dirty[dirty == "999"] <- NA
```

then, you can coerce the data into the types they should be

```
dirty$time <- as.Date(dirty$time,'%m/%d/%Y')
dirty$dept <- as.factor(dirty$dept)
dirty$enroll <- as.factor(dirty$enroll)
dirty$birth.order <- as.numeric(dirty$birth.order)
```

```
## Warning: NAs introduced by coercion
```

```
str(dirty)
```

```
## 'data.frame':    5 obs. of  5 variables:
##  $ time       : Date, format: "2015-07-25" "2015-07-25" ...
##  $ height     : chr  "very" "70" "5’9" "2.1" ...
##  $ dept       : Factor w/ 2 levels "anthro","geology": 2 NA 2 2 1
##  $ enroll     : Factor w/ 2 levels "No","Yes": 2 2 NA 1 NA
##  $ birth.order: num  1 1 2 NA 2
```

your turn!

I’ve intentionally left the height variable alone. Take a look at it now. What happened here?

2. Missingness

introduction

there are many reasons why you might have missing data

AS LONG AS MISSINGNESS IS NOT CAUSED BY YOUR INDEPENDENT VARIABLE this is fine

deleting those observations is wasteful, but easy (listwise deletion)

ignoring the individual missing data points is typical (casewise deletion)

imputing mean values for missing data is possibly the worst thing you can do

imputing via MI + error is currently the best option

listwise deletion is wasteful

```
na.omit(dirty)
```

```
##         time height    dept enroll birth.order
## 1 2015-07-25   very geology    Yes           1
```

casewise deletion is what R does internally

```r
nrow(dirty)
sum(is.na(dirty$height))
sum(is.na(dirty$birth.order))
length(lm(height ~ birth.order, data=dirty)$fitted.values)
```

this is usually the default strategy

remember how we talked about the extensibility of R?

amelia is a package that makes a complicated MI approach work without you knowing anything about its implementation

```r
library(Amelia)
```

```
## Loading required package: Rcpp
## ##
## ## Amelia II: Multiple Imputation
## ## (Version 1.7.3, built: 2014-11-14)
## ## Copyright (C) 2005-2015 James Honaker, Gary King and Matthew Blackwell
## ## Refer to http://gking.harvard.edu/amelia/ for more information
## ##
```

let's use this large dataset as an example

```r
large <- read.csv('data/large.csv')
summary(large)
```

```
##        a                    b               c
##  Min.   :-33.98426   Min.   :-13.4   Min.   :-249998.64
##  1st Qu.: -6.71903   1st Qu.:128.6   1st Qu.:-141005.65
##  Median :  0.41681   Median :256.9   Median : -63498.56
##  Mean   :  0.00176   Mean   :252.2   Mean   : -83954.09
##  3rd Qu.:  7.00630   3rd Qu.:377.5   3rd Qu.: -15748.98
##  Max.   : 35.33306   Max.   :513.3   Max.   :     11.77
##  NA's   :45          NA's   :45      NA's   :45
```

```r
nrow(na.omit(large))
```

```
## [1] 871
```

for it to work you need low missingness and large N

```r
a <- amelia(large,m = 1)
```

```
## -- Imputation 1 --
##
##   1  2  3
```

```r
print(a)
```

```
##
## Amelia output with 1 imputed datasets.
## Return code:  1
## Message:  Normal EM convergence.
##
## Chain Lengths:
## --------------
## Imputation 1:  3
```

amelia returns a list, where the first item is a list of your imputations

we only did one, so here it is

```r
large.imputed <- a[[1]][[1]]
summary(large.imputed)
```

```
##        a                   b                c
##  Min.   :-33.98426   Min.   :-13.4   Min.   :-249999
##  1st Qu.: -6.60227   1st Qu.:128.4   1st Qu.:-140069
##  Median :  0.39075   Median :252.1   Median : -63513
##  Mean   : -0.00721   Mean   :250.4   Mean   : -83286
##  3rd Qu.:  6.94988   3rd Qu.:373.9   3rd Qu.: -15626
##  Max.   : 35.33306   Max.   :567.7   Max.   :  70966
```

if you give it a tiny dataset, it will fuss at you

```r
a <- amelia(large[990:1000,],m = 1)
```

```
## Warning in amelia.prep(x = x, m = m, idvars = idvars, empri = empri, ts =
## ts, : You have a small number of observations, relative to the number, of
## variables in the imputation model. Consider removing some variables, or
## reducing the order of time polynomials to reduce the number of parameters.

## -- Imputation 1 --
##
##   1  2
```

```r
print(a)
```

```
##
## Amelia output with 1 imputed datasets.
## Return code:  1
## Message:  Normal EM convergence.
##
## Chain Lengths:
## --------------
## Imputation 1:  2
```

your turn!

imagine I'm interested in measuring the partial pressure of oxygen on academic performance, and I get these data:

```
oxygen <- data.frame(kPa = c(0, 10, 20, 30, 40), test = c(NA, NA, 90, 95, NA))
oxygen <- oxygen[sample(nrow(oxygen), 1000, replace=TRUE), ]
```

can I use amelia on this dataset? how should you fix this?

3. Tidyness

introduction

now that our data is clean, it's time to put it in a tidy format. this is a way of storing data that makes it easy to:

1. make graphs
2. run tests
3. summarize
4. transform into other formats

we are basically trying to organize ourselves such that:

1. any grouping is made on rows
2. any testing is done between columns

an aside on testing

in R, you use double symbols for testing

```
1 == 2
```

```
## [1] FALSE
```

```
1 != 1
```

```
## [1] FALSE
```

```
1 >= 1
```

```
## [1] TRUE
```

(you've already seen a couple of these)

tests return boolean vectors

```
1 >= c(0,1,2)
```

```
## [1]  TRUE  TRUE FALSE
```

recall that boolean vectors need to be the same length or a divisor

if your vectors are not multiples of each other, R will fuss at you

```
c(1,2) >= c(1,2,3)
```

```
## Warning in c(1, 2) >= c(1, 2, 3): longer object length is not a multiple of
## shorter object length

## [1]  TRUE  TRUE FALSE
```

```
c(1,2) >= c(1,2,3,4)      # why no warning this time? R recycles!
```

```
## [1]  TRUE  TRUE FALSE FALSE
```

the combination of the length requirement, the lack of support in R for proper indexing, and missingness in your data will cause many headaches later on

subsetting data frames

subsetting your data is where you will use this regularly

```
dirty$birth.order == 2
```

```
## [1] FALSE FALSE  TRUE    NA  TRUE
```

```
dirty[dirty$birth.order == 2, ]
```

```
##          time height    dept enroll birth.order
## 3  2015-07-25    5'9 geology   <NA>           2
## NA       <NA>   <NA>    <NA>   <NA>          NA
## 5  2015-07-25    156  anthro   <NA>           2
```

you can also select columns

```
dirty[ ,'dept']
```

```
## [1] geology <NA>    geology geology anthro
## Levels: anthro geology
```

that empy space **before** the comma? that tells R to grab all the rows

you can also match elements from a vector

```
good.things <- c("geology", "anthro")
dirty[dirty$dept %in% good.things, ]
```

```
##         time height    dept enroll birth.order
## 1 2015-07-25   very geology    Yes           1
## 3 2015-07-25    5'9 geology   <NA>           2
## 4 2015-07-25    2.1 geology     No          NA
## 5 2015-07-25    156  anthro   <NA>           2
```

most tidying can be done with two R packages

(plus a wrapper around the base string functions)

```
library(reshape2)
library(stringr)
library(plyr)
```

tidyness

our goal here is to arrange our data such that each table is about one kind of thing: whether it is everything about a measurement, everything about a person, or everything about a group of people

```
abnormal <- data.frame(name = c('Alice','Bob','Eve'),
                       time1 = c(90,90,150),
                       time2 = c(100,95,100))
```

this table is not tidy - why not?

the table is about measurements, but each measurement does not have its own row, and each type of measurement value is represented by more than one column

`melt` takes wide frames and makes them long

```
normal <- melt(data = abnormal, id.vars = 'name')
normal
```

```
##    name variable value
## 1 Alice    time1    90
## 2   Bob    time1    90
## 3   Eve    time1   150
## 4 Alice    time2   100
## 5   Bob    time2    95
## 6   Eve    time2   100
```

we can `melt` this dataframe down into a long format, which makes each row a unique observation, and then clean up the dataframe a bit

```r
normal$id <- seq(1:nrow(normal))
names(normal) <- c('name','time','value','id')
normal$time <- str_replace(normal$time,'time','')
```

subsetting tidy data is easy

now that we are in a tidy format, see how easy it is to subset

```r
normal[normal$time == 1,]
```

```
##    name time value id
## 1 Alice    1    90  1
## 2   Bob    1    90  2
## 3   Eve    1   150  3
```

```r
normal[normal$name == 'Alice',]
```

```
##    name time value id
## 1 Alice    1    90  1
## 4 Alice    2   100  4
```

and test

```r
t.test(value ~ time, data=normal)
```

```
##
##  Welch Two Sample t-test
##
## data:  value by time
## t = 0.58132, df = 2.0278, p-value = 0.6191
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -73.56101  96.89434
## sample estimates:
## mean in group 1 mean in group 2
##       110.00000        98.33333
```

join tidy dataframes with `merge`

imagine you have two datasets that you want to merge

```r
data.1 <- read.csv('data/merge_practice_1.csv')
data.2 <- read.csv('data/merge_practice_2.csv')
```

```
## Warning in read.table(file = file, header = header, sep = sep, quote
## = quote, : incomplete final line found by readTableHeader on 'data/
## merge_practice_2.csv'
```

```r
str(data.1)
```

```
## 'data.frame':    5 obs. of  4 variables:
##  $ id      : int  1 2 3 4 5
##  $ name    : Factor w/ 5 levels "Alice","Bob",..: 1 2 3 4 5
##  $ job     : Factor w/ 3 levels "communications",..: 1 1 2 1 3
##  $ location: Factor w/ 3 levels "Berkeley","Cambridge",..: 3 2 3 1 2
```

```r
str(data.2)
```

```
## 'data.frame':    4 obs. of  4 variables:
##  $ id      : int  1 4 5 6
##  $ name    : Factor w/ 4 levels "Alice","Dave",..: 1 2 3 4
##  $ job     : Factor w/ 3 levels "hacker","handler",..: 1 3 2 1
##  $ location: Factor w/ 4 levels "berkeley","cambridge",..: 2 4 3 1
```

sometimes the same people have differet jobs in different locations

you can do an *inner* join using merge

```r
merge(data.1, data.2, by = 'id')
```

```
##   id name.x          job.x location.x name.y   job.y location.y
## 1  1  Alice communications   New York  Alice  hacker  cambridge
## 2  4   Dave communications   Berkeley   Dave    tree  palo alto
## 3  5    Eve            spy  Cambridge    Eve handler   new york
```

that's no good - we lost half of our people!

inner joins are mostly used when you **only** want records that appear in both tables

if you want the union, you can use an outer join

```r
merge(data.1, data.2, by = 'id', all = TRUE)
```

```
##   id name.x          job.x location.x name.y   job.y location.y
## 1  1  Alice communications   New York  Alice  hacker  cambridge
## 2  2    Bob communications  Cambridge   <NA>    <NA>       <NA>
## 3  3  Chuck         hacker   New York   <NA>    <NA>       <NA>
## 4  4   Dave communications   Berkeley   Dave    tree  palo alto
## 5  5    Eve            spy  Cambridge    Eve handler   new york
## 6  6   <NA>           <NA>       <NA>  Faith  hacker   berkeley
```

this works basically the same as `join` in SQL

your turn!

running merges is particularly useful when:

a. your data is tidy; and,
b. you want to add information with a lookup table

in this case, you can store your lookup table as a dataframe, then merge it

```
lookup <- read.csv('data/merge_practice_3.csv')
str(lookup)
```

```
## 'data.frame':    5 obs. of  2 variables:
##  $ location  : Factor w/ 5 levels "Berkeley","Cambridge",..: 2 3 1 4 5
##  $ population: int  107289 8406000 116768 66642 233294
```

how would you merge these?

look at the third table - there is data for the population of Reno, NV - why doesn't this show up in the merged table?

4. Transforming data

introduction

because R started out as a functional language, it can be hard to modify data, especially in place

in practice, if you want 100% control over how your frames are being modified, you'll be writing lots of `for` loops, which is messy

luckily, there is a package that handles the common tasks for you

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
##
## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
##
## The following objects are masked from 'package:stats':
##
##     filter, lag
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

sort data with `arranage`

base R syntax for sorting is a bit of a pain in that you have to create a sorting vector based on the values in a column, then subset the same dataframe and apply the sorting vector to the rows slice

to demonstrate this, let's start by making a toy data frame

```
toy <- data.frame(
  id = c(1,1,1,2,2,2,3,3,3),
  score.1 = c(90,94,40,80,80,80,76,80,82)
)
arrange(toy, score.1)
```

```
##   id score.1
## 1  1      40
## 2  3      76
## 3  2      80
## 4  2      80
## 5  2      80
## 6  3      80
## 7  3      82
## 8  1      90
## 9  1      94
```

select rows by pattern with `select`

it's common for variables that measure similar things to have similar names, but selecting columns this was in base R requires running `grepl` on column names, then subsetting the dataframe and applying the logical vector to the column field

```
toy$score.2 <- 100
select(toy, score.1, score.2)
```

```
##   score.1 score.2
## 1      90     100
## 2      94     100
## 3      40     100
## 4      80     100
## 5      80     100
## 6      80     100
## 7      76     100
## 8      80     100
## 9      82     100
```

```
select(toy, contains('score'))
```

```
##   score.1 score.2
## 1      90     100
## 2      94     100
## 3      40     100
## 4      80     100
## 5      80     100
```

```
## 6      80     100
## 7      76     100
## 8      80     100
## 9      82     100
```

apply summary fucntions with `summarise`

dplyr includes most of the base R summary statistics, along with:

- `n()`
- `n_distinct()`
- `first()`
- `last()`

```r
summarise(toy, n(), n_distinct(score.1), last(score.1))
```

```
##   n() n_distinct(score.1) last(score.1)
## 1   9                   6            82
```

dplyr allows you to apply functions to groups

so far, these have taken base R functions and made them faster (with C++ calls behind the scenes), easier to use, or both

dplyr's real utility is in its grouped dataframes, which apply dplyr functions groupwise

```r
group_by(toy, id)
```

```
## Source: local data frame [9 x 3]
## Groups: id
##
##   id score.1 score.2
## 1  1      90     100
## 2  1      94     100
## 3  1      40     100
## 4  2      80     100
## 5  2      80     100
## 6  2      80     100
## 7  3      76     100
## 8  3      80     100
## 9  3      82     100
```

```r
summarise(group_by(toy, id), n(), n_distinct(score.1))
```

```
## Source: local data frame [3 x 3]
##
##   id n() n_distinct(score.1)
## 1  1   3                   3
## 2  2   3                   1
## 3  3   3                   3
```

you can add as many functions as you want inbetween, but wrapping function call around function call can be hard to read (and write!)

you can pipe functions with the %>% operator

this will look very familiar if you are used to working in bash

```
toy %>% group_by(id) %>% summarise(n(), n_distinct(score.1))
```

```
## Source: local data frame [3 x 3]
##
##   id n() n_distinct(score.1)
## 1  1   3                   3
## 2  2   3                   1
## 3  3   3                   3
```

your turn!

take another look at the D-Lab training feedback dataset, and see if you can use this grouping, selecting, and summarizing syntax to find out which department gives the highest average ratings

imagine that you wanted to divide each rating by its department average - could you do this using dplyr and merge?

5. Descriptive statistics

introduction

data analysis generally procedes in two steps:

1. exploratory data analysis (now)
2. statistical inference (tomorrow)

our treatment of exploratory analysis owes a lot to John Tukey and to the Grammar of Graphics

let's load in some data about D-Lab feedback

```
load('data/feedback.Rda')
str(dat)
```

```
## 'data.frame':    1062 obs. of  14 variables:
##  $ timestamp              : Date, format: "2015-04-23" "2015-04-23" ...
##  $ course.delivered       : int  7 7 7 6 7 6 3 6 5 7 ...
##  $ instructor.communicated: int  6 7 5 6 7 6 2 4 4 7 ...
##  $ hear                   : Factor w/ 51 levels "-","a colleague",..: 19 19 19 34 13 NA 24 19 24 31
##  $ interest               : int  7 7 7 6 6 7 6 7 7 7 ...
##  $ department             : Factor w/ 27 levels "African American Studies",..: NA NA NA NA NA NA NA
##  $ verbs                  : chr  "This was a helpful workshop. \n\nKelly was a clear instructor and
##  $ useful                 : int  7 7 7 6 6 6 3 7 4 7 ...
##  $ gender                 : Factor w/ 3 levels "Female/Woman",..: 2 2 NA 1 1 2 2 NA 1 1 ...
##  $ ethnicity              : chr  "Asian American" "White" "White" "White" ...
```

```
##  $ outside.barriers       : int  2 1 1 3 1 1 1 NA 1 1 ...
##  $ inside.barriers        : int  1 1 1 1 1 1 1 NA 1 1 ...
##  $ what.barriers          : chr  NA NA NA NA ...
##  $ position               : Factor w/ 23 levels "Academic staff title",..: 20 4 4 4 9 2 14 NA 15 20
```

R provides two easy/simple summary functions in the base package

```
summary(dat)
```

```
##    timestamp          course.delivered instructor.communicated
##  Min.   :2014-08-19   Min.   :1.000    Min.   :1.000
##  1st Qu.:2014-11-05   1st Qu.:6.000    1st Qu.:6.000
##  Median :2015-01-30   Median :7.000    Median :7.000
##  Mean   :2015-01-22   Mean   :6.251    Mean   :6.257
##  3rd Qu.:2015-04-03   3rd Qu.:7.000    3rd Qu.:7.000
##  Max.   :2015-06-22   Max.   :7.000    Max.   :7.000
##
##                                     hear        interest
##  Email from the D-Lab mailing list     :340   Min.   :1.0
##  Found it on the D-Lab website         :278   1st Qu.:6.0
##  Heard about it from a friend/colleague:247   Median :7.0
##  Email from another mailing list       : 99   Mean   :6.6
##  Don't remember                        : 12   3rd Qu.:7.0
##  (Other)                               : 55   Max.   :7.0
##  NA's                                  : 31   NA's   :15
##               department     verbs               useful
##  Public Health      : 81   Length:1062        Min.   :1.00
##  Public Policy      : 44   Class :character   1st Qu.:5.00
##  Sociology          : 38   Mode  :character   Median :6.00
##  Political Science  : 36                      Mean   :6.02
##  Integrative Biology: 28                      3rd Qu.:7.00
##  (Other)            :288                      Max.   :7.00
##  NA's               :547
##                               gender     ethnicity
##  Female/Woman                    :579   Length:1062
##  Male/Man                        :332   Class :character
##  Genderqueer/Gender non-conforming:  1   Mode  :character
##  NA's                            :150
##
##
##
##  outside.barriers inside.barriers what.barriers
##  Min.   :1.000    Min.   :1.000   Length:1062
##  1st Qu.:1.000    1st Qu.:1.000   Class :character
##  Median :1.000    Median :1.000   Mode  :character
##  Mean   :2.073    Mean   :1.259
##  3rd Qu.:3.000    3rd Qu.:1.000
##  Max.   :5.000    Max.   :5.000
##  NA's   :167      NA's   :175
##                             position
##  PhD student, dissertation stage: 41
##  PhD student, pre-dissertation  : 33
```

```
##  Visiting fellow or researcher  : 24
##  Masters student                : 22
##  Undergraduate student          : 21
##  (Other)                        : 64
##  NA's                           :857
```

```r
table(dat$department)
```

```
##
##  African American Studies  Ag & Resource Econ & Pol
##                        24                        23
##              Anthropology   App Sci & Tech Grad Grp
##                        12                        10
##    Biostatistics Grad Grp City & Regional Planning
##                         8                        20
##                 Economics                 Education
##                        23                        26
##  Energy & Resources Group   Env Sci, Policy, & Mgmt
##                        14                        17
##   Ethnic Studies Grad Grp                   History
##                         1                        17
## Industrial Eng & Ops Rsch               Information
##                         4                         9
##       Integrative Biology              JSP Grad Pgm
##                        28                         6
##                       Law               Linguistics
##                         9                        11
##                     Music              Neuroscience
##                         3                         4
##         Political Science                Psychology
##                        36                        28
##             Public Health             Public Policy
##                        81                        44
##                  Rhetoric    Slavic Languages & Lit
##                        11                         8
##                 Sociology
##                        38
```

think back to day one - how would we make weekdays out of the date variable?

```r
dat$wday <- factor(weekdays(dat$timestamp, abbreviate = TRUE),
                   levels = c('Mon','Tue','Wed','Thu','Fri','Sat','Sun')
                   )
summary(dat$wday)
```

```
## Mon Tue Wed Thu Fri Sat Sun
## 168 124 144 323 277  16  10
```

reshape provides a few more ways to aggregate things

```r
library(reshape2)
dcast(dat[dat$gender == 'Female/Woman' | dat$gender == 'Male/Man',], department ~ gender)
```

```
## Using wday as value column: use value.var to override.
## Aggregation function missing: defaulting to length
```

```
##                     department Female/Woman Male/Man  NA
## 1   African American Studies            8       16   0
## 2   Ag & Resource Econ & Pol           20        3   0
## 3                Anthropology           9        3   0
## 4    App Sci & Tech Grad Grp            6        4   0
## 5     Biostatistics Grad Grp            5        3   0
## 6   City & Regional Planning           12        7   0
## 7                  Economics           16        5   0
## 8                  Education           20        3   0
## 9   Energy & Resources Group           10        3   0
## 10   Env Sci, Policy, & Mgmt           11        5   0
## 11   Ethnic Studies Grad Grp            1        0   0
## 12                   History            9        6   0
## 13 Industrial Eng & Ops Rsch            2        2   0
## 14               Information            2        7   0
## 15       Integrative Biology           20        8   0
## 16               JSP Grad Pgm           5        1   0
## 17                       Law            5        4   0
## 18               Linguistics            8        1   0
## 19                     Music            2        0   0
## 20              Neuroscience            0        4   0
## 21         Political Science           17       18   0
## 22                Psychology           20        8   0
## 23             Public Health           55       19   0
## 24             Public Policy           22       21   0
## 25                  Rhetoric            0       11   0
## 26    Slavic Languages & Lit            7        1   0
## 27                 Sociology           23       12   0
## 28                      <NA>          264      157 150
```

```r
dcast(melt(dat, measure.vars = c('course.delivered')), wday ~ 'Delivered', fun.aggregate = mean)
```

```
##   wday Delivered
## 1  Mon  6.309524
## 2  Tue  6.274194
## 3  Wed  6.159722
## 4  Thu  6.077399
## 5  Fri  6.444043
## 6  Sat  6.250000
## 7  Sun  6.600000
```

your turn!

imagine you are interested in whether opinions about D-Lab vary based on academic position - how would you make a table about this?

6. Plotting

every time you use `base::plot`, Edward Tufte does something unkind to a cute animal

- we'll be using ggplot, R's implementation of the **grammar of graphics**
- in this grammar, you use 'aesthetics' to define how data is mapped to objects the graph space
- each graph space has at least three layers:
  - theme/background/annotations
  - axes
  - objects
- most objects are geometric shapes
- some objects are statistics built on those shapes
- you can stack as many layers as you like

```
install.packages('ggplot2')
```

```
##
## The downloaded binary packages are in
##  /var/folders/rj/8gpcssqd52z9yrqw7f8xxfym0000gn/T//RtmpZiaJXk/downloaded_packages
```

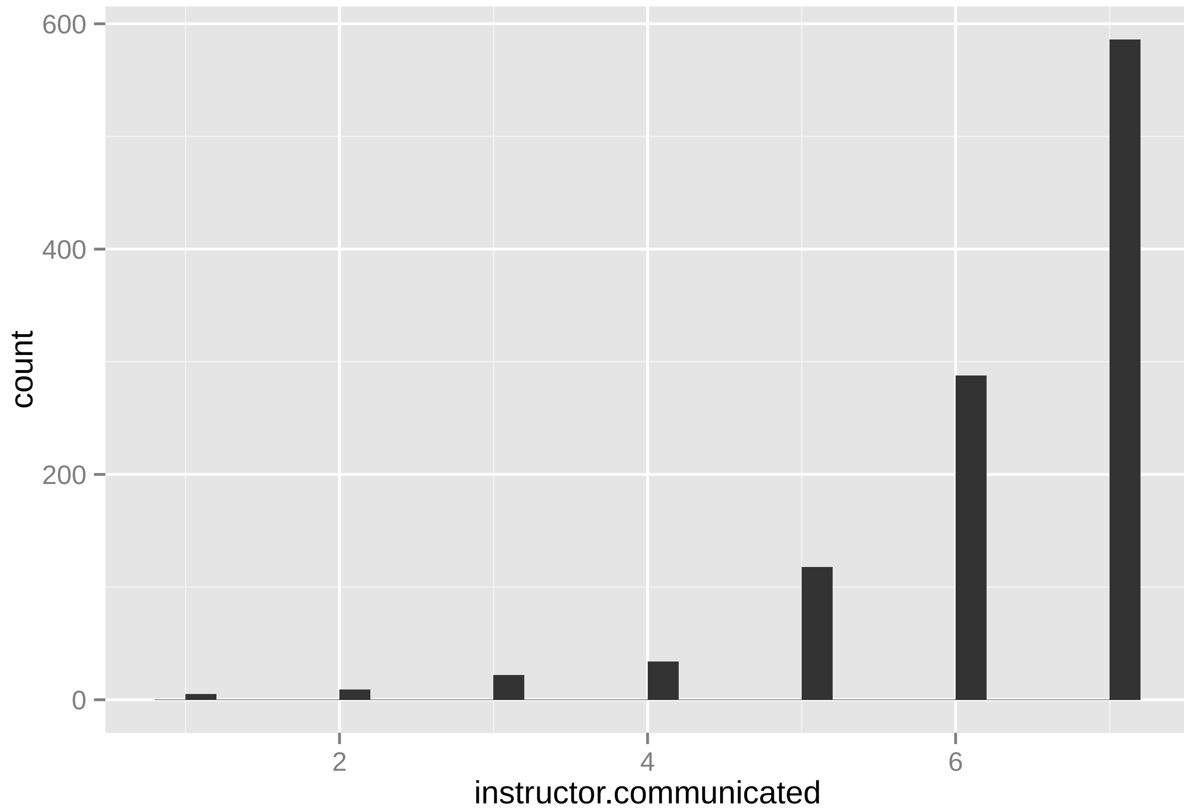
```
library(ggplot2)
```

use qplot for initial poking around

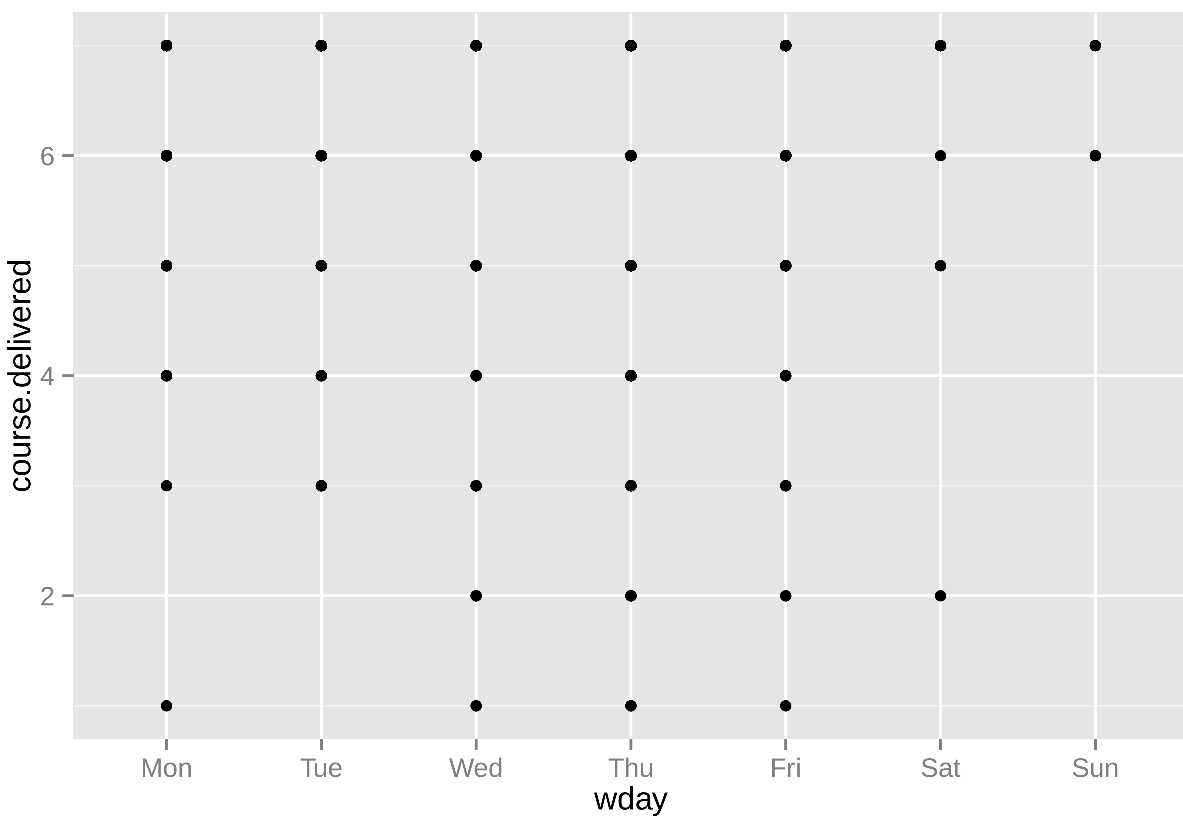
it has very strong intuitions about what you want to see, and is not particularly customizable

```
qplot(instructor.communicated, data = dat)
```

```
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```

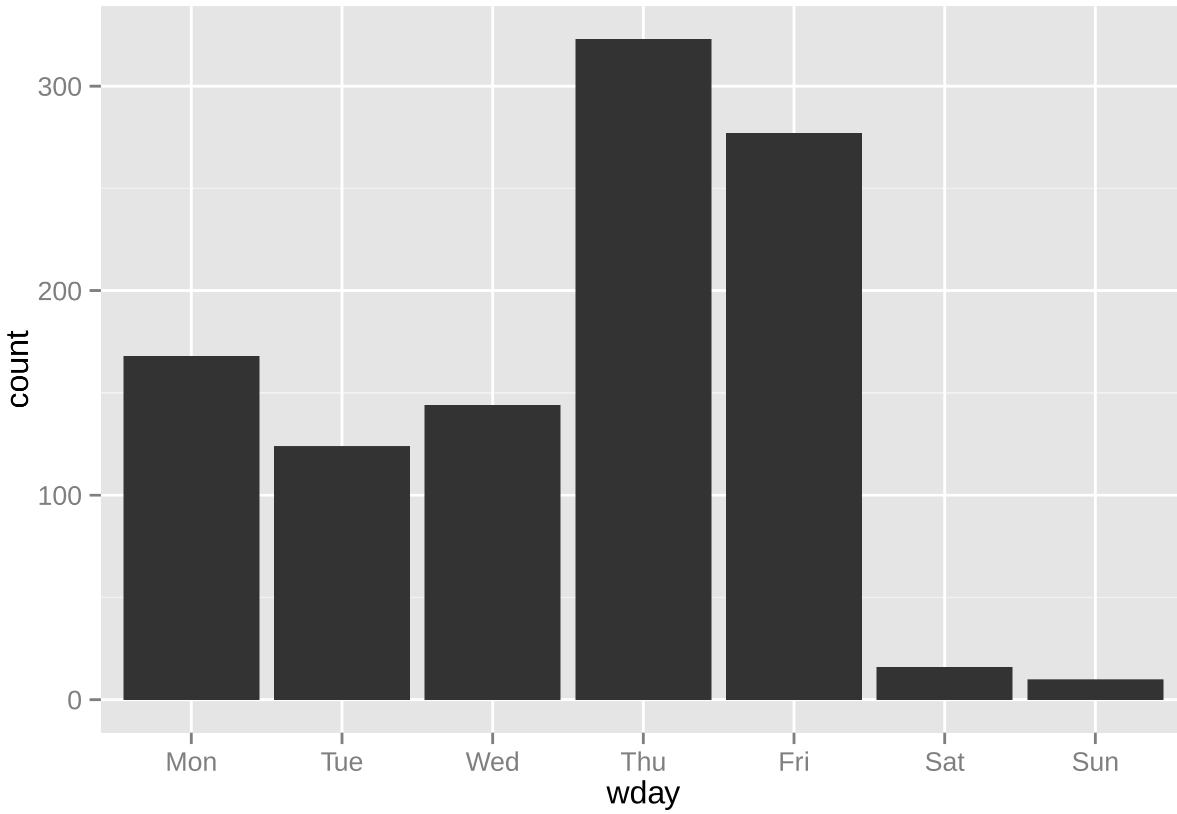



```
qplot(wday, course.delivered, data = dat)
```



for 1D cateforical, use bar

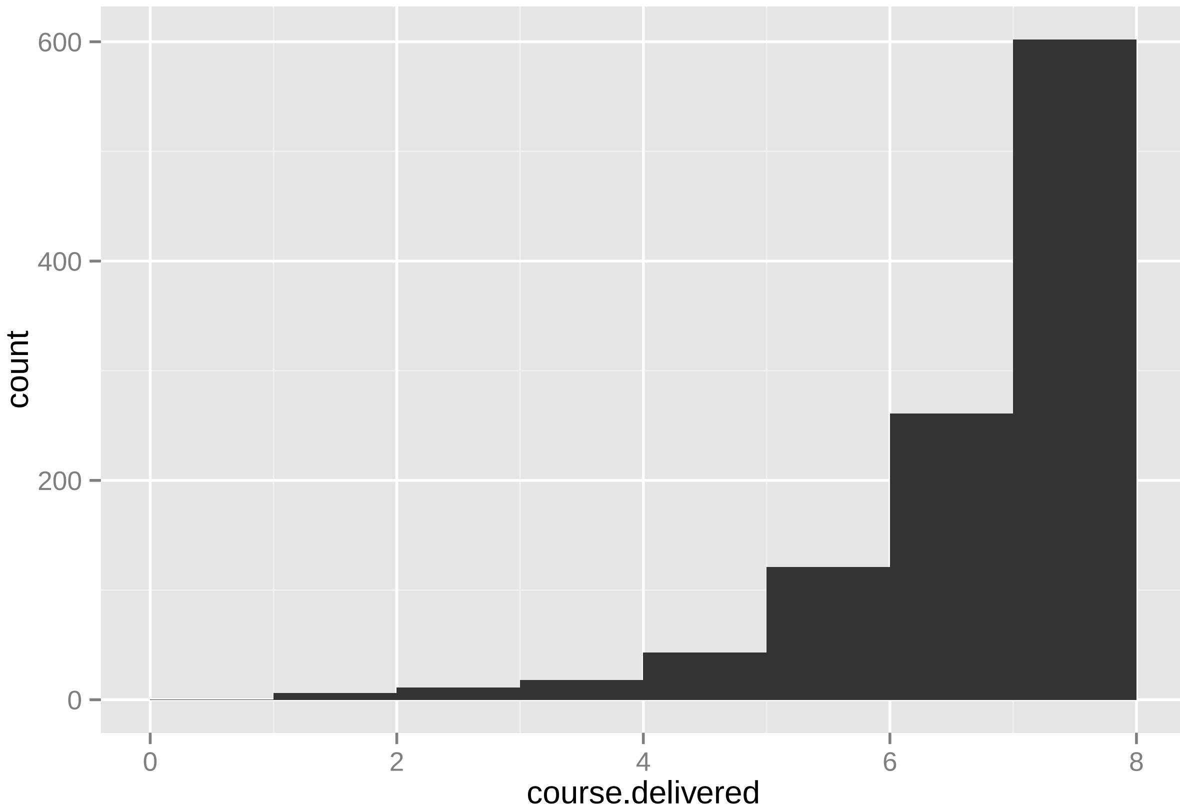
```
ggplot(data=dat, aes(x=wday)) + geom_bar()
```



for 1D continuous, use hist

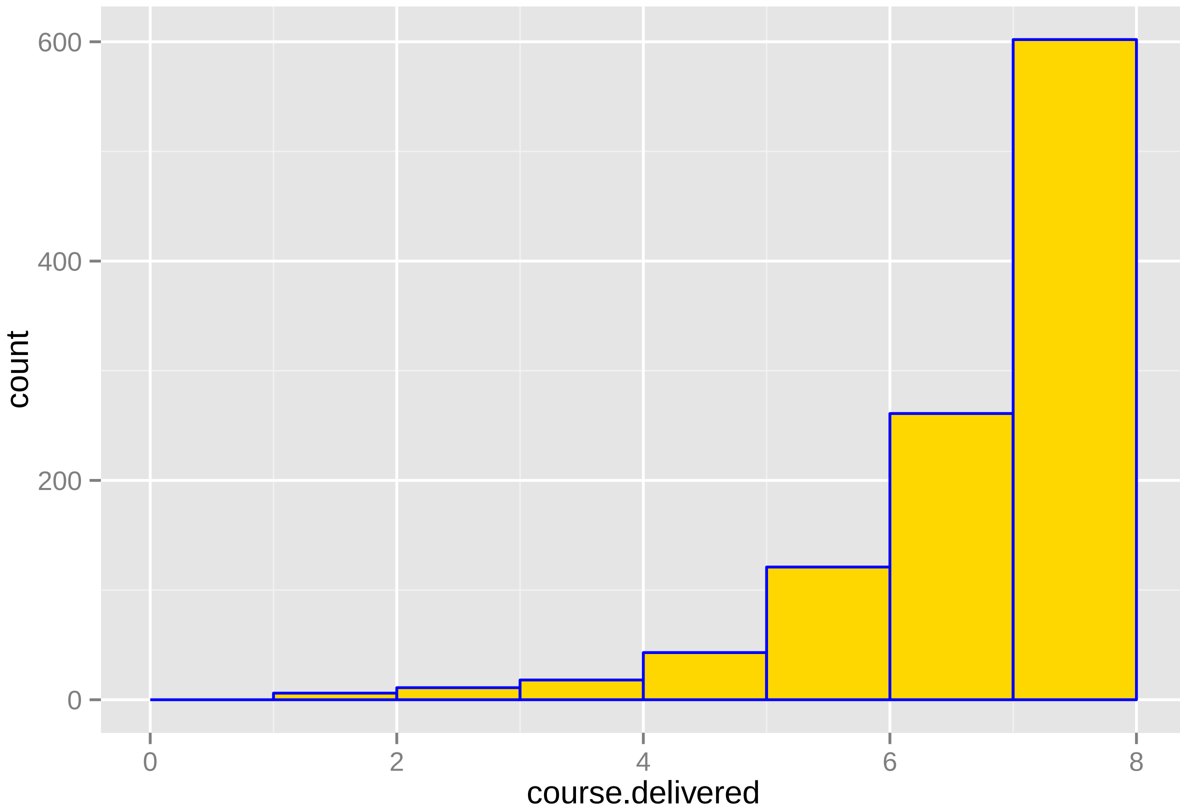
this is really just convenience for `geom_bar(stat = 'bin')`, as opposed to bar plots, whose `stat` is `'count'`

```
ggplot(data=dat, aes(x=course.delivered)) +
  geom_histogram(binwidth=1)
```



you can add color to this plot

```
ggplot(data=dat, aes(x=course.delivered)) +
  geom_histogram(binwidth=1, fill = 'gold', colour= 'blue')
```



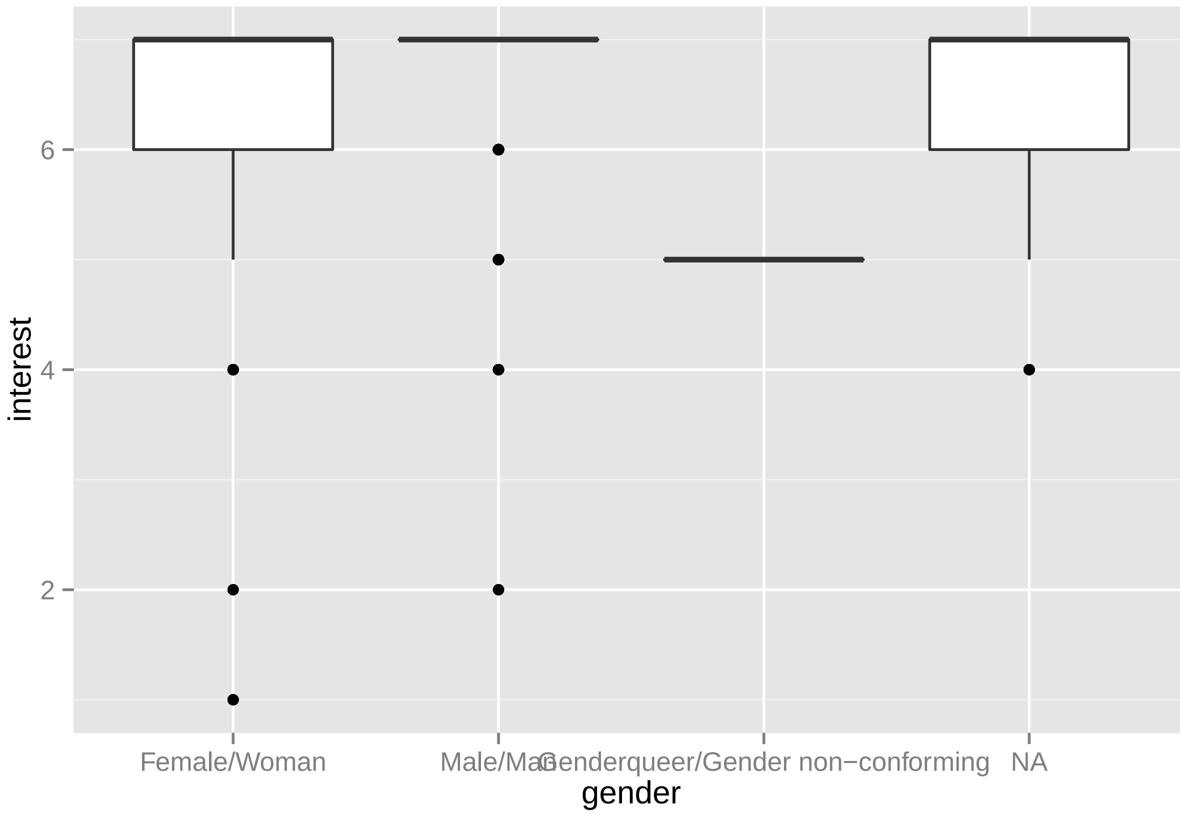
GO BEARS

for many 1D variables, use a box plot

these are handy for a whole bunch of reasons, and you should make them your close associates

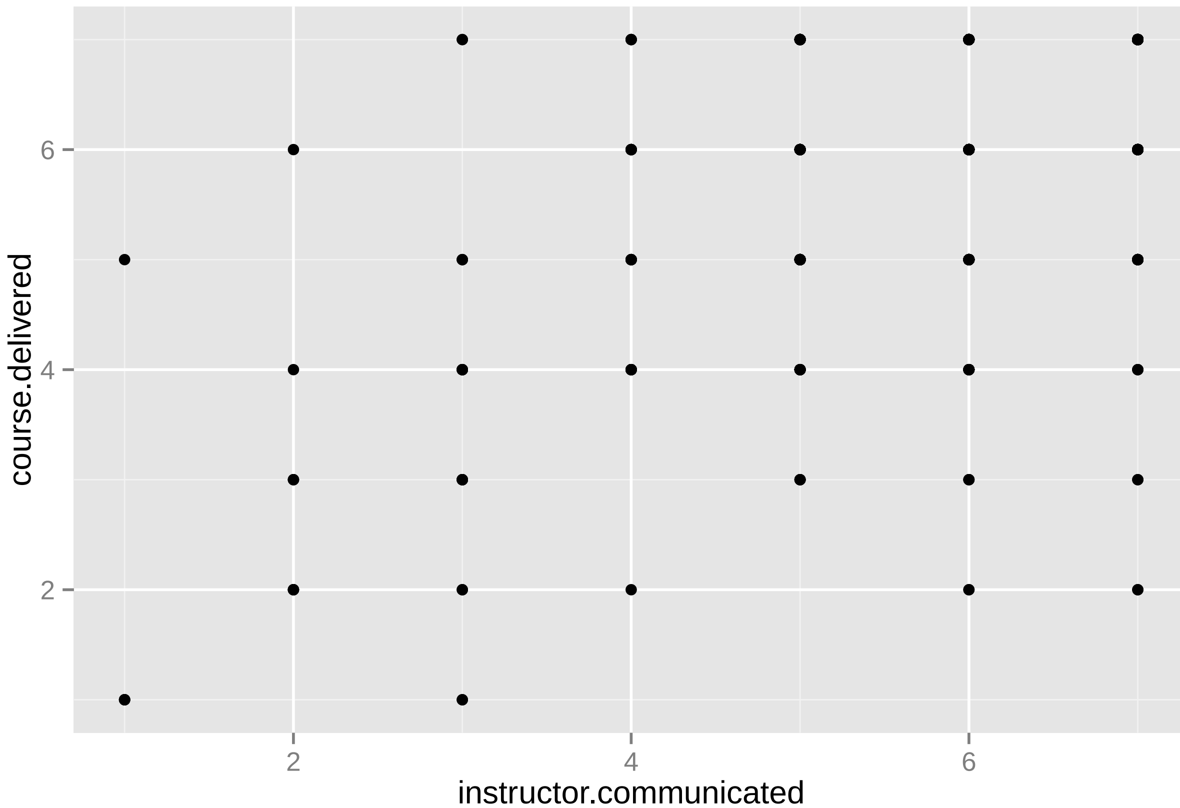
```
ggplot(data=dat, aes(x=gender,y=interest)) + geom_boxplot()
```

```
## Warning: Removed 15 rows containing non-finite values (stat_boxplot).
```



to plot two continuous variables, use points

```
ggplot(data=dat, aes(x=instructor.communicated, y=course.delivered)) + geom_point()
```

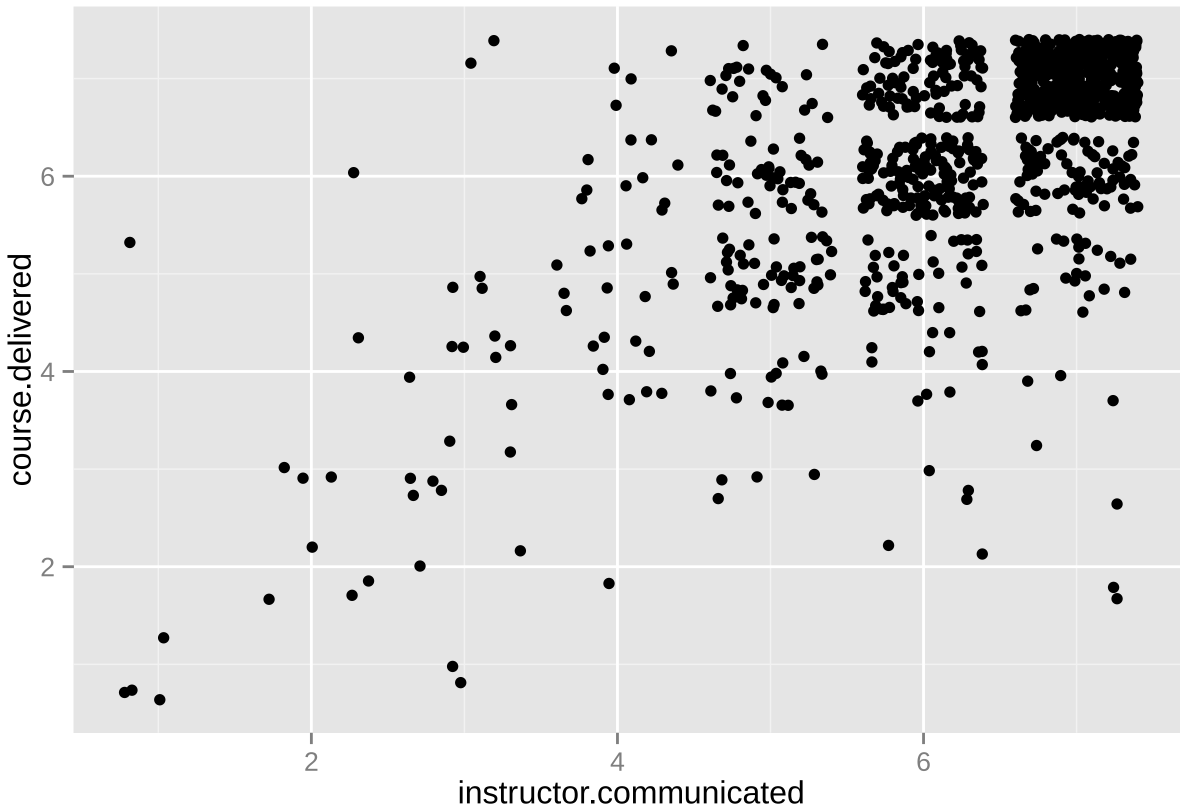


all of these values are discrete, which makes them hard to see

to scatter points randomy, use jitter

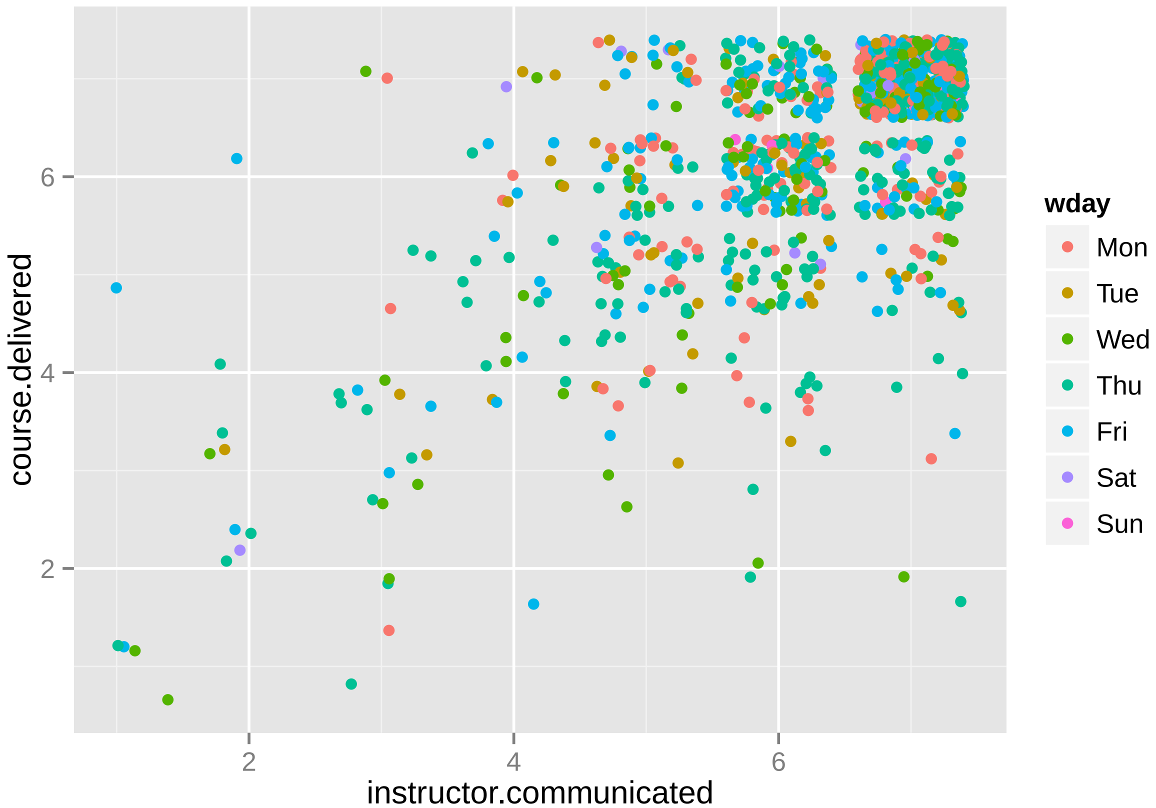
this is really just convenience for `geom_point(position = jitter())`

```
ggplot(data=dat, aes(x=instructor.communicated, y=course.delivered)) +
  geom_jitter()
```



not only can you add color, you can make the color a mapping of other variables

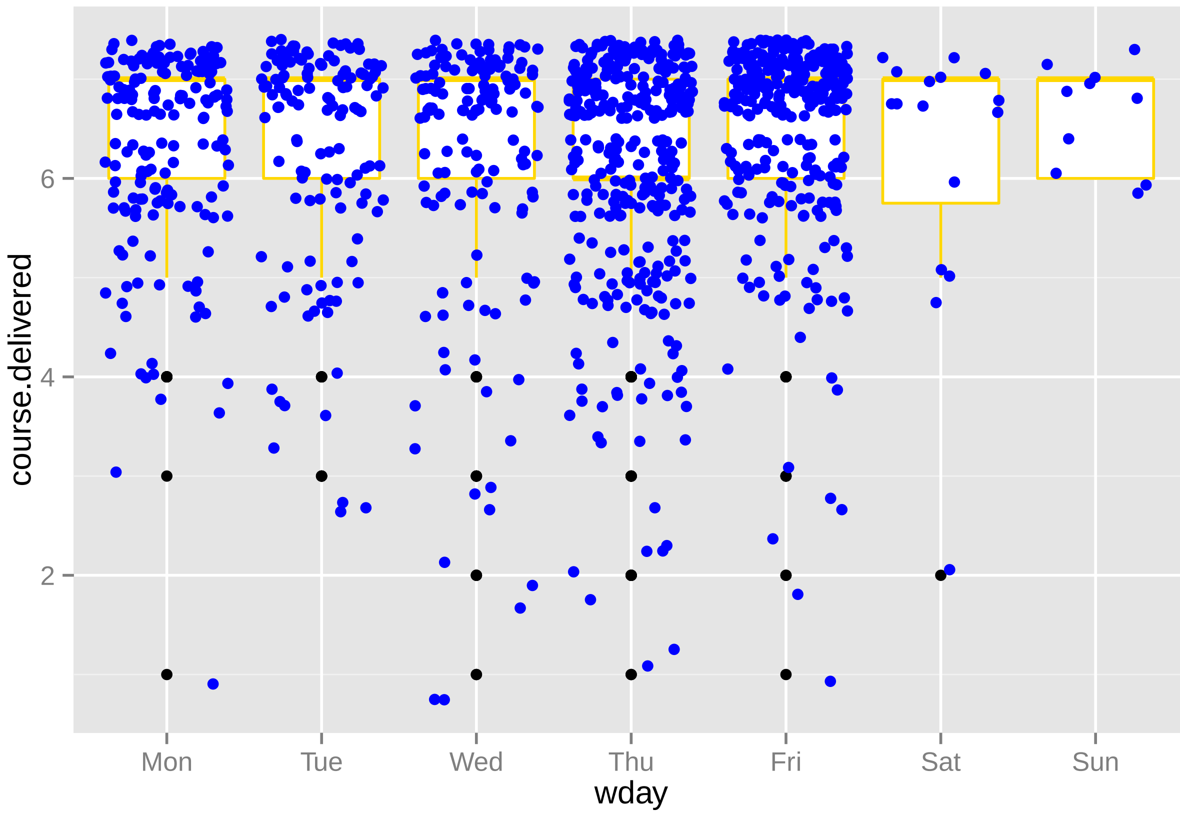
```
ggplot(data=dat, aes(x=instructor.communicated, y=course.delivered)) +
  geom_jitter(aes(colour = wday))
```



the last time we used `colour` it was not an aesthetic - why is it now?

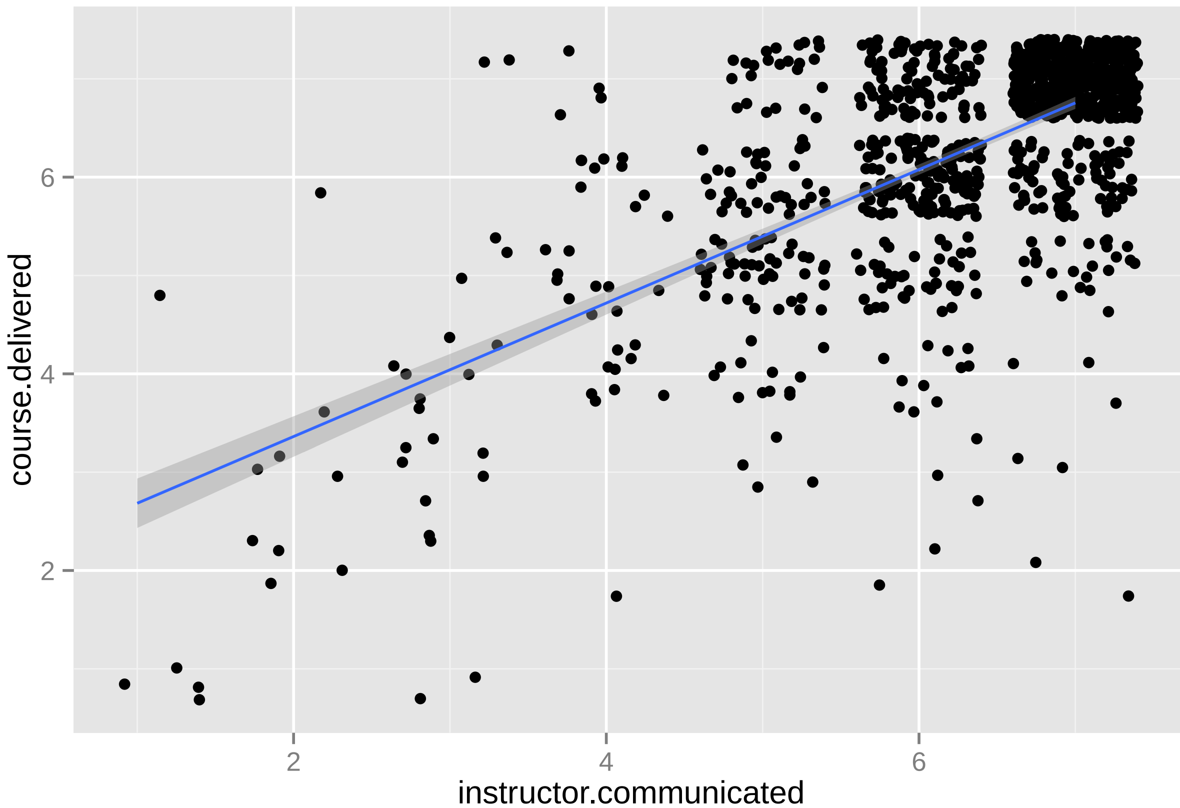
you can stack layers until your eyes hurt

```
ggplot(data=dat, aes(x=wday, y=course.delivered)) +
  geom_boxplot(colour = 'gold') +
  geom_jitter(colour = 'blue')
```



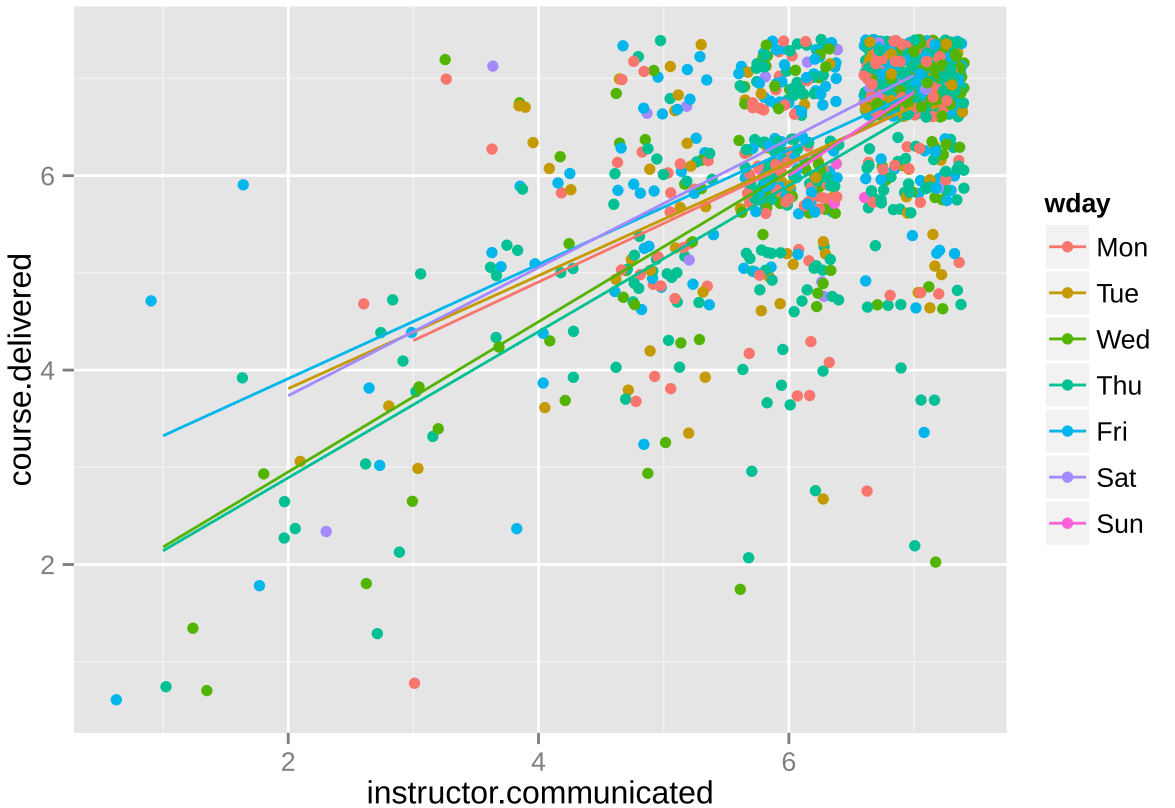
add summary functions with smooth

```
ggplot(data=dat, aes(x=instructor.communicated, y=course.delivered)) +
  geom_jitter() +
  stat_smooth(method = 'lm')
```



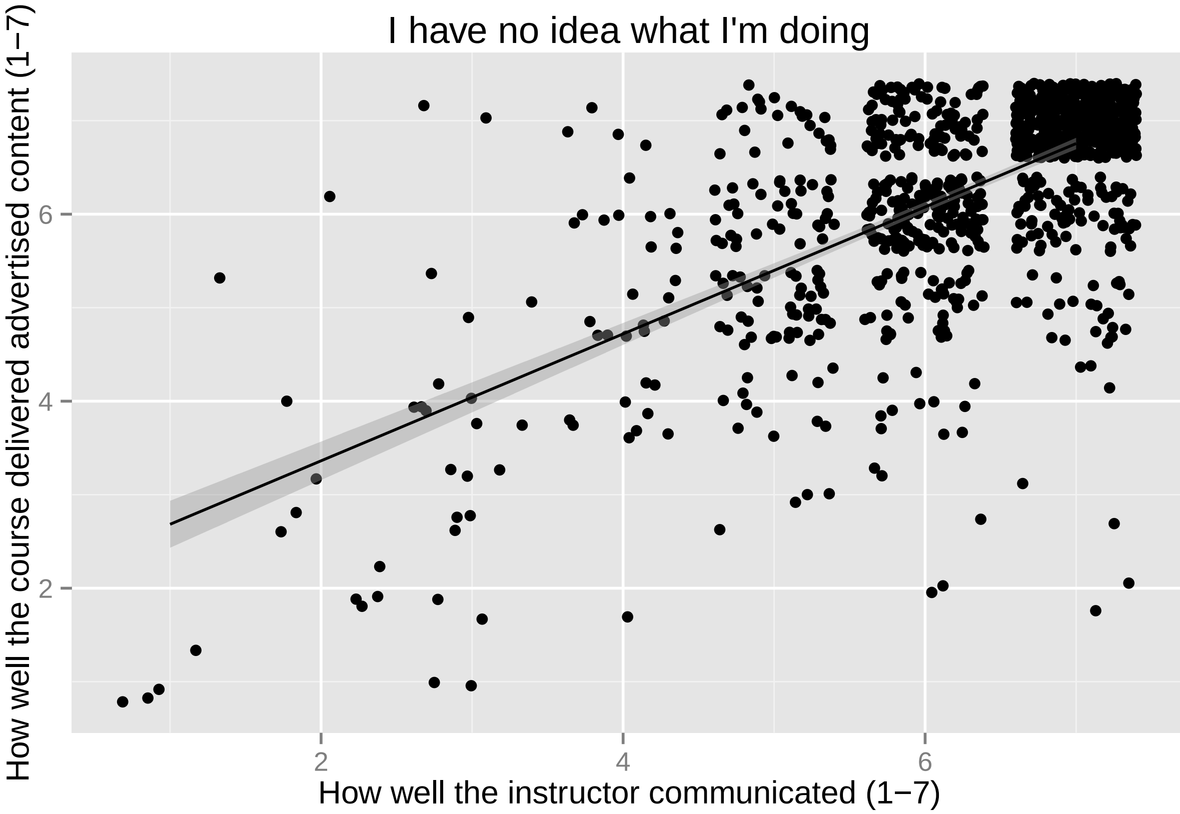
if you are using colour as an aesthetic, you'll produce stats for each color

```
ggplot(data=dat, aes(x=instructor.communicated, y=course.delivered, colour = wday)) +
  geom_jitter() +
  stat_smooth(method = 'lm', se = FALSE)
```



good scientists put units on their axes

```
ggplot(data=dat, aes(x=instructor.communicated, y=course.delivered)) +
  geom_jitter() +
  stat_smooth(method = 'lm', colour = 'black') +
  xlab('How well the instructor communicated (1-7)') +
  ylab('How well the course delivered advertised content (1-7)') +
  ggtitle("I have no idea what I'm doing")
```



the general point here is that every single object on this graph is customizable

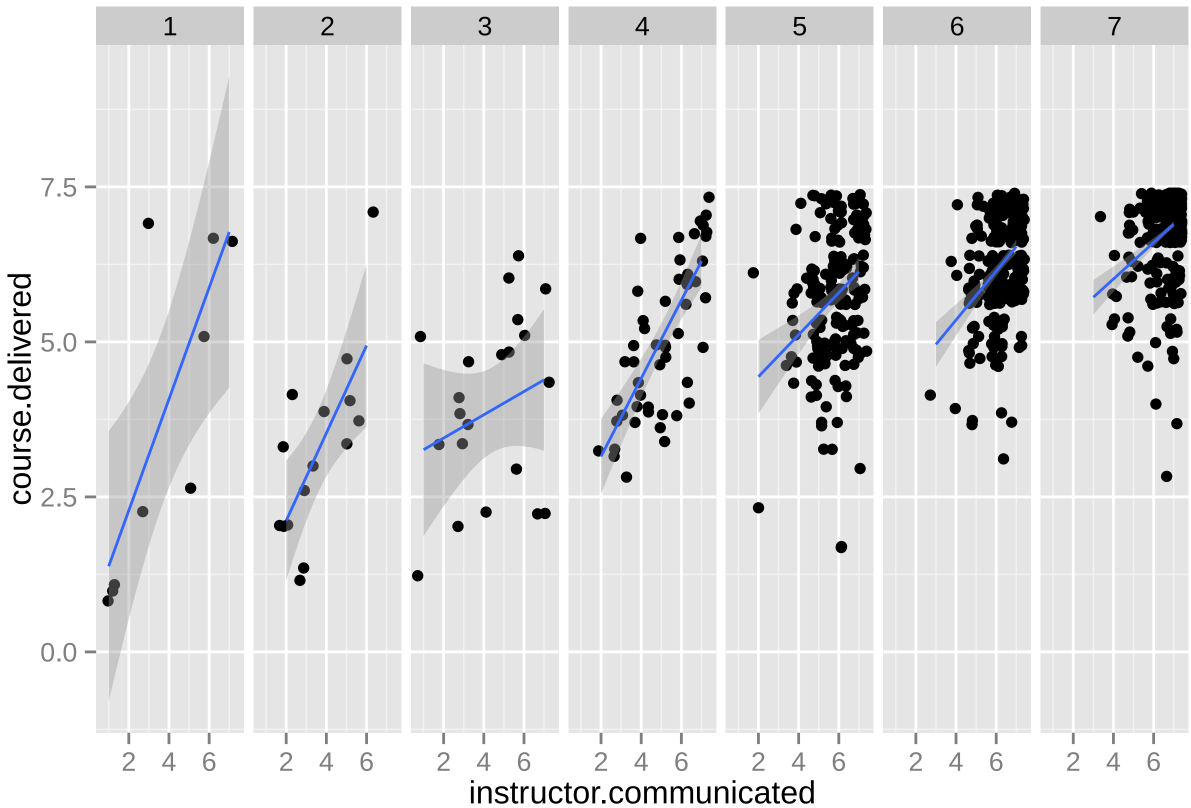
frequent customizations are very simple to add

infrequent customizations will take a lot of tinkering on your part

facetting

often useful for looking at relationships between three variables at the same time

```
ggplot(data=dat, aes(x=instructor.communicated, y=course.delivered)) +
  geom_jitter() +
  stat_smooth(method = 'lm') +
  facet_grid(. ~ useful)
```



your turn!

There were a lot of variables in this dataset that we did not look at today:

```
names(data)
```

```
## NULL
```

Choose two of those variables, and explore their distribution and relationship to each other. Can you conclude anything about the D-Lab based on the feedback?