Lecture 7

Brad McNeney

Topics

- Tidying, or reshaping data
- Split-apply-combine

Tidying (Reshaping) Data

Tidying data with the tidyverse

```
library(tidyverse) # loads dplyr, ggplot2, tidyr, etc
```

- Reading: Hadley Wickham's article on tidy data, available from GitHub (Notes folder) or canvas.
  - Today's notes closely follow the `tidyr` vignette.
- "Tidy" data is ready for analysis, with one row for each sampled unit and columns for the variables measured on the units.
  - Often classify variables as "explanatory" or "response"
- Tabular data and repeated measures data are often not in tidy form.
- Examples:
  - Tabular data on new tuberculosis cases from WHO
  - Repeated measures from Billboard Top 100

Tuberculosis (TB) cases

```
tb <- read.csv("tb.csv",stringsAsFactors=FALSE)
dim(tb)
```

```
## [1] 5769   23
```

```
tb[1:4,1:6]
```

```
##   iso2 year new_sp new_sp_m04 new_sp_m514 new_sp_m014
## 1   AD 1989     NA          NA           NA          NA
## 2   AD 1990     NA          NA           NA          NA
## 3   AD 1991     NA          NA           NA          NA
## 4   AD 1992     NA          NA           NA          NA
```

```
names(tb)[1:20]
```

```
##  [1] "iso2"          "year"          "new_sp"        "new_sp_m04"
##  [5] "new_sp_m514"   "new_sp_m014"   "new_sp_m1524" "new_sp_m2534"
##  [9] "new_sp_m3544" "new_sp_m4554" "new_sp_m5564" "new_sp_m65"
## [13] "new_sp_mu"     "new_sp_f04"    "new_sp_f514"   "new_sp_f014"
## [17] "new_sp_f1524" "new_sp_f2534" "new_sp_f3544" "new_sp_f4554"
```

Structure of TB table

- First column is 2-letter country code, second is year, third is number of new cases for that country/year.
- Then come TB counts for different gender/age categories.
  - `new_sp` is "new cases by positive pulmonary smear assay"
  - gender is `m` or `f`
  - two special age categories 0-4, 5-14,
  - age categories 0-14, 15-24, 25-34, 35-44, 45-54, 55-65, 65+, unknown (`u`)
- Gender/age columns are not variables, they are data on the observed units.
- Tidy data would have one row for each country, year, gender and age category, with a column of counts

Billboard Top 100 rankings of songs

```
bb <- read.csv("billboard.csv",stringsAsFactors = FALSE)
dim(bb)
```

```
## [1] 317  83
```

```
bb[1:3,1:6]
```

```
##   year artist.inverted                    track time genre date.entered
## 1 2000 Destiny's Child Independent Women Part I 3:38  Rock   2000-09-23
## 2 2000         Santana             Maria, Maria 4:18  Rock   2000-02-12
## 3 2000   Savage Garden       I Knew I Loved You 4:07  Rock   1999-10-23
```

```
names(bb)[c(1:10,ncol(bb))]
```

```
##  [1] "year"            "artist.inverted" "track"
##  [4] "time"            "genre"           "date.entered"
##  [7] "date.peaked"     "x1st.week"       "x2nd.week"
## [10] "x3rd.week"       "x76th.week"
```

Structure of the Billboard table

- Columns `year` through `date.peaked` describe the song, then `x1st.week` through `x76th.week` are the chart positions for the first through 76th weeks.
  - If a song is on the chart for less than 76 weeks, its position is `NA` for any missing weeks.
- Weeks are not variables, they are the time data for the time series.

Tidying the Billboard data

- Main step is to consolidate, or "gather" the rankings in the different weeks into a `rank` variable.
- Before gathering, will select/rename some of the variables.
- After gathering, will create some new variables and sort the data frame.

Select and rename

- Won't need `time` or `genre`.
  - `select()` from `dplyr` can use `-` to de-select
- Rename `artist.inverted`
  - `rename()` from `dplyr` takes arguments of the form `newname = oldname`

```
bb <-
  bb %>% select(-time,-genre) %>%
  rename(artist = artist.inverted)
```

Gather the weeks into a "long" version of the Billboard data

- Leave each song info variable as-is.
- The data, or "values", are the chart positions.
- The weeks are descriptors or "keys" for these values.
- We want to create key-value pairs for each observation.
  - There will be missing values, which we can remove.
- The `gather()` function from `tidyr` gathers specified columns into keys (e.g., week) and values (e.g., rank).

gather() for the Billboard data

```
# gather (data, key, value, ... ) where ... are the columns to collapse
bblong <- gather(bb,week,rank,x1st.week:x76th.week,na.rm=TRUE)
head(bblong,n=4)
```

```
##   year          artist                    track date.entered date.peaked
## 1 2000 Destiny's Child Independent Women Part I   2000-09-23  2000-11-18
## 2 2000         Santana             Maria, Maria   2000-02-12  2000-04-08
## 3 2000   Savage Garden       I Knew I Loved You   1999-10-23  2000-01-29
## 4 2000         Madonna                    Music   2000-08-12  2000-09-16
##        week rank
## 1 x1st.week   78
## 2 x1st.week   15
## 3 x1st.week   71
## 4 x1st.week   41
```

More cleaning suggested in the vignette

- Extract week numbers from `week` variable
- Coerce `date.entered` to a `Date` object
- Calculate the date of each ranking based on the date it entered the charts and the week.
- Sort ("arrange") on artist, track and week.

```
bb <-
  bblong %>% mutate(week = parse_number(week),
                date = as.Date(date.entered) + 7*(week-1)) %>%
  select(-date.entered) %>% # don't need date.entered anymore
  arrange(artist,track,week)
head(bb,n=3)
```

```
##   year artist                                  track date.peaked week rank
## 1 2000  2 Pac Baby Don't Cry (Keep Ya Head Up II)  2000-03-11    1   87
## 2 2000  2 Pac Baby Don't Cry (Keep Ya Head Up II)  2000-03-11    2   82
## 3 2000  2 Pac Baby Don't Cry (Keep Ya Head Up II)  2000-03-11    3   72
##         date
## 1 2000-02-26
## 2 2000-03-04
## 3 2000-03-11
```

Tidying the TB data

- Recall structure of the data: country, year, count of new cases, counts of new cases by gender/age categories.

```
names(tb)[1:10]
```

```
##  [1] "iso2"          "year"          "new_sp"        "new_sp_m04"
##  [5] "new_sp_m514"   "new_sp_m014"   "new_sp_m1524"  "new_sp_m2534"
##  [9] "new_sp_m3544"  "new_sp_m4554"
```

- Main step is to "gather" TB prevalence in the different gender/age categories into a count variable.
  - Complicated by the coding of gender/age categories
- Before gathering, will remove unneeded variables and add country names to supplement 2-letter codes.

Remove variables

- Won't need overall count
- Special categories 0-4 and 5-14 overlap with 0-14 so remove
- Age unknown not useful for analysing trends, so remove

```
tb <- select(tb,-new_sp, -contains("04"), -contains("514"),
             -new_sp_mu, -new_sp_fu)
tb[1:3,1:10]
```

```
##   iso2 year new_sp_m014 new_sp_m1524 new_sp_m2534 new_sp_m3544
## 1   AD 1989          NA            NA            NA            NA
## 2   AD 1990          NA            NA            NA            NA
## 3   AD 1991          NA            NA            NA            NA
##   new_sp_m4554 new_sp_m5564 new_sp_m65 new_sp_f014
## 1           NA           NA          NA          NA
## 2           NA           NA          NA          NA
## 3           NA           NA          NA          NA
```

Add country names to supplement country codes

- I found a translation of the ISO-2 country codes at [http://data.okfn.org/data/core/country-list] and saved as `countryCodes.csv` in the `Notes` folder.

```
cc <- read.csv("countryCodes.csv",stringsAsFactors = FALSE)
# cc has columns "Name" and "Code". "Code" matches "iso2" in tb
```

- Exercise: Find out which ISO-2 codes are in `tb` but not in `countryCodes.csv`, google the missing codes, and add the country names to `cc` manually.

```
unique(tb$iso2[!(tb$iso2 %in% cc$Code)])
```

```
## [1] "AN" "YU"
```

```
cc <- rbind(cc, data.frame(Name=c("Netherlands Antilles","Yugoslavia"),
                           Code=c("AN","YU")))
tb <- inner_join(cc,tb,by = c("Code" = "iso2"))
tb[1:2,1:6]
```

```
##          Name Code year new_sp_m014 new_sp_m1524 new_sp_m2534
## 1 Afghanistan   AF 1980           NA            NA           NA
## 2 Afghanistan   AF 1981           NA            NA           NA
```

Gather counts for demographic groups

- Create demographic variable `demog` and count variable `count` by gathering over all variables except `Name`, `Code` and `year`.

```
tblong <- gather(tb,demog,count,-Name,-Code,-year,na.rm=TRUE)
head(tblong)
```

```
##             Name Code year       demog count
## 13 Afghanistan   AF 1997 new_sp_m014     0
## 14 Afghanistan   AF 1998 new_sp_m014    30
## 15 Afghanistan   AF 1999 new_sp_m014     8
## 16 Afghanistan   AF 2000 new_sp_m014    52
## 17 Afghanistan   AF 2001 new_sp_m014   129
## 18 Afghanistan   AF 2002 new_sp_m014    90
```

Separate gender from age category.

- First remove `new_sp_`, then separate result on first column (`help(separate)`)

```
maxlen <- max(nchar(tblong$demog))
tblong %>% mutate(demog = substr(demog,8,maxlen)) %>%
  separate(demog, into=c("gender","agecat"),sep=1) -> tb
head(tb)
```

```
##          Name Code year gender agecat count
## 1 Afghanistan   AF 1997      m    014     0
## 2 Afghanistan   AF 1998      m    014    30
## 3 Afghanistan   AF 1999      m    014     8
## 4 Afghanistan   AF 2000      m    014    52
## 5 Afghanistan   AF 2001      m    014   129
## 6 Afghanistan   AF 2002      m    014    90
```

```
save(tb,file="tb.RData")
```

Split-Apply-Combine

Subgroup summaries

- Data visualization and modelling is often in terms of subgroups.
- Illustrate with some data on enrollments in Stat and Act Sci courses over the 2007/08 to 2015/16 academic years.
  - Data on full-time equivalents (FTEs, equal to 30 credit hours taught) by year and course.
- Recurring theme: Need to split the data into subgroups, transform or summarize, and reassemble, or unsplit.
  - Has come to be known as "split-apply-combine"

Science enrollments database

- Load the `scilong` data frame created by `FTE.Rmd`
  - Look through the `FTE.Rmd` script if you haven't already.

```
library(tidyverse)
load("scilong.RData")
head(scilong)
```

```
##   Subject CrsNum CreditHrs semester enrollment     FTEs year
## 1    ACMA    210         3     1077         51  5.10000 2008
## 2    ACMA    335         3     1077         20  2.00000 2008
## 3    ACMA    425         3     1077         22  2.20000 2008
## 4    ACMA    465         3     1077         16  1.60000 2008
## 5    ACMA    490         3     1077          4  0.40000 2008
## 6    BISC    100         4     1077        176 23.46667 2008
```

Stat and Act Sci data

```
stat <- filter(scilong,Subject=="STAT" | Subject=="ACMA")
head(stat)
```

```
##   Subject CrsNum CreditHrs semester enrollment FTEs year
## 1    ACMA    210         3     1077         51  5.1 2008
## 2    ACMA    335         3     1077         20  2.0 2008
## 3    ACMA    425         3     1077         22  2.2 2008
## 4    ACMA    465         3     1077         16  1.6 2008
## 5    ACMA    490         3     1077          4  0.4 2008
## 6    STAT    100         3     1077         49  4.9 2008
```

Split-apply-combine example 1: yearly percent FTEs

- Suppose we want the percent of FTEs in a year that are attributable to each course taught.
- Split the data by year, compute proportion of FTEs for each course in that year, and combine the proportions into a variable that can be included in the `stat` data frame.
- Illustrate base R and `dplyr` approaches.

Example 1: split

- The base R function `split()` splits a data frame on a grouping variable, which is a vector or list of vectors that can be coerced to factor(s), and returns a list.

```
sp.stat <- split(stat,stat$year)
names(sp.stat)
```

```
## [1] "2008" "2009" "2010" "2011" "2012" "2013" "2014" "2015" "2016"
```

```
head(sp.stat[["2008"]])
```

```
##   Subject CrsNum CreditHrs semester enrollment FTEs year
## 1    ACMA    210         3     1077         51  5.1 2008
## 2    ACMA    335         3     1077         20  2.0 2008
## 3    ACMA    425         3     1077         22  2.2 2008
## 4    ACMA    465         3     1077         16  1.6 2008
## 5    ACMA    490         3     1077          4  0.4 2008
## 6    STAT    100         3     1077         49  4.9 2008
```

```
str(sp.stat[["2008"]])
```

```
## 'data.frame':    47 obs. of  7 variables:
##  $ Subject   : chr  "ACMA" "ACMA" "ACMA" "ACMA" ...
```

Example 1: Split, cont.

```
sp.stat <- split(stat,list(stat$year,stat$Subject))
names(sp.stat)
```

```
##  [1] "2008.ACMA" "2009.ACMA" "2010.ACMA" "2011.ACMA" "2012.ACMA"
##  [6] "2013.ACMA" "2014.ACMA" "2015.ACMA" "2016.ACMA" "2008.STAT"
## [11] "2009.STAT" "2010.STAT" "2011.STAT" "2012.STAT" "2013.STAT"
## [16] "2014.STAT" "2015.STAT" "2016.STAT"
```

```
head(sp.stat[["2008.STAT"]])
```

```
##    Subject CrsNum CreditHrs semester enrollment FTEs year
## 6     STAT    100         3     1077         49  4.9 2008
## 7     STAT    101         3     1077         59  5.9 2008
## 8     STAT    201         3     1077        284 28.4 2008
## 9     STAT    203         3     1077        164 16.4 2008
## 10    STAT    270         3     1077        185 18.5 2008
## 11    STAT    285         3     1077         47  4.7 2008
```

group_by() from dplyr

- Call is similar to `split`, but we specify multiple variables to group on by comma-separated names.
- Output is a `tibble` (data frame with some different default behaviours).

```
sp.stat.dplyr <- group_by(stat,year,Subject)
sp.stat.dplyr
```

```
## # A tibble: 468 x 7
## # Groups:   year, Subject [18]
##    Subject CrsNum CreditHrs semester enrollment  FTEs  year
##    <chr>   <chr>      <int>    <dbl>      <int> <dbl> <dbl>
##  1 ACMA    210            3     1077         51   5.1  2008
##  2 ACMA    335            3     1077         20   2    2008
##  3 ACMA    425            3     1077         22   2.2  2008
##  4 ACMA    465            3     1077         16   1.6  2008
##  5 ACMA    490            3     1077          4   0.4  2008
##  6 STAT    100            3     1077         49   4.9  2008
##  7 STAT    101            3     1077         59   5.9  2008
##  8 STAT    201            3     1077        284  28.4  2008
##  9 STAT    203            3     1077        164  16.4  2008
## 10 STAT    270            3     1077        185  18.5  2008
## # ... with 458 more rows
```

Example 1: Apply

- Create a new variable `FTEproportion = FTEs/sum(FTEs)` for each sub-group data frame and save the new variable in the respective data frames.
- Can use the base R function `lappy()`
  - stands for "list apply" – apply a function to each element of a list and return a list as output
- It turns out the following call to `lapply()` does what we want.

```
tem <- lapply(sp.stat,transform,FTEproportion=FTEs/sum(FTEs))
```

- To see why, start with simpler uses of `lapply()`.

Simpler example of `lapply()`

- Define a function to apply to each list element and apply it:

```
fsum <- function(x) { # x is a list element
  sum(x$FTEs) # assumes list elements have an FTEs column
}
tem <- lapply(sp.stat,fsum)
tem[1:2]
```

```
## $`2008.ACMA`
## [1] 20.36667
##
## $`2009.ACMA`
## [1] 18.63333
```

Simpler example, cont.

- If our function takes more arguments than just the list element, we add them after the function name.

```
fsum <- function(x,cname) {
  sum(x[,cname])
}
tem <- lapply(sp.stat,fsum,"FTEs")
tem[1:2]
```

```
## $`2008.ACMA`
## [1] 20.36667
##
## $`2009.ACMA`
## [1] 18.63333
```

Our use of `lapply()`

- Adding a column to each sub-group data frame requires a function that takes the data frame as an argument and returns the augmented version.
  - This is what `transform()` does

```
head(transform(sp.stat[[1]],FTEproportion = FTEs/sum(FTEs)))
```

```
##    Subject CrsNum CreditHrs semester enrollment FTEs year FTEproportion
## 1     ACMA    210         3     1077         51  5.1 2008    0.25040917
## 2     ACMA    335         3     1077         20  2.0 2008    0.09819967
## 3     ACMA    425         3     1077         22  2.2 2008    0.10801964
## 4     ACMA    465         3     1077         16  1.6 2008    0.07855974
## 5     ACMA    490         3     1077          4  0.4 2008    0.01963993
## 19    ACMA    315         3     1081         21  2.1 2008    0.10310966
```

Putting it all together

```
sp.stat <- lapply(sp.stat,transform,FTEproportion=FTEs/sum(FTEs))
head(sp.stat[[1]])
```

```
##    Subject CrsNum CreditHrs semester enrollment FTEs year FTEproportion
## 1     ACMA    210         3     1077         51  5.1 2008    0.25040917
## 2     ACMA    335         3     1077         20  2.0 2008    0.09819967
## 3     ACMA    425         3     1077         22  2.2 2008    0.10801964
## 4     ACMA    465         3     1077         16  1.6 2008    0.07855974
## 5     ACMA    490         3     1077          4  0.4 2008    0.01963993
## 19    ACMA    315         3     1081         21  2.1 2008    0.10310966
```

Detour: The apply family of functions in R

- The "original" apply is `apply()`, which can be used to apply a function to rows or columns of a matrix.

```
mat <- matrix(1:6,ncol=2,nrow=3)
mat
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```
apply(mat,1,sum) # row-wise sums; rowSums() is faster
```

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

```
apply(mat,2,sum) # column-wise; colSums() is faster
```

```
## [1]  6 15
```

Detour, cont.

- `sapply()` takes the output of `lapply()` and simplifies to a vector or matrix.

```
fsum <- function(x) { sum(x$FTEs) }
sapply(sp.stat,fsum)[1:2]
```

```
## 2008.ACMA 2009.ACMA
##  20.36667  18.63333
```

Detour, cont.

- Other `apply`-like functions `vapply()`, `mapply()`, `tapply()`, ...
- I don't use these.
  - See their respective help pages for information.

The apply step with dplyr

- Actions ("verbs") like `mutate()` are applied to the data within groups when passed a grouped object.
  - That is, the data table is broken into groups and `mutate()` is applied separately to each group.

```
sp.stat.dplyr <- mutate(sp.stat.dplyr,FTEpp = FTEs/sum(FTEs))
select(sp.stat.dplyr,Subject,FTEs,year,FTEpp)
```

```
## # A tibble: 468 x 4
## # Groups:   year, Subject [18]
##    Subject  FTEs  year  FTEpp
##    <chr>   <dbl> <dbl>  <dbl>
##  1 ACMA      5.1  2008 0.250
##  2 ACMA      2    2008 0.0982
##  3 ACMA      2.2  2008 0.108
##  4 ACMA      1.6  2008 0.0786
##  5 ACMA      0.4  2008 0.0196
##  6 STAT      4.9  2008 0.0208
##  7 STAT      5.9  2008 0.0250
##  8 STAT     28.4  2008 0.121
##  9 STAT     16.4  2008 0.0696
## 10 STAT     18.5  2008 0.0785
## # ... with 458 more rows
```

The combine step

- The base R function `unsplit()` will combine the elements of the list that was generated by `split()`
- Pass `unsplit()` the list of variables used to define the splits.

```
head(unsplit(sp.stat,list(stat$year,stat$Subject)))
```

```
##   Subject CrsNum CreditHrs semester enrollment FTEs year FTEproportion
## 1    ACMA    210         3     1077         51  5.1 2008    0.25040917
## 2    ACMA    335         3     1077         20  2.0 2008    0.09819967
## 3    ACMA    425         3     1077         22  2.2 2008    0.10801964
## 4    ACMA    465         3     1077         16  1.6 2008    0.07855974
## 5    ACMA    490         3     1077          4  0.4 2008    0.01963993
## 6    STAT    100         3     1077         49  4.9 2008    0.02079796
```

The combine step with dplyr

- Use ungroup()

```
ungroup(sp.stat.dplyr)
```

```
## # A tibble: 468 x 8
##    Subject CrsNum CreditHrs semester enrollment  FTEs  year  FTEpp
##    <chr>   <chr>      <int>    <dbl>      <int> <dbl> <dbl>  <dbl>
##  1 ACMA    210            3     1077         51   5.1  2008 0.250
##  2 ACMA    335            3     1077         20   2    2008 0.0982
##  3 ACMA    425            3     1077         22   2.2  2008 0.108
##  4 ACMA    465            3     1077         16   1.6  2008 0.0786
##  5 ACMA    490            3     1077          4   0.4  2008 0.0196
##  6 STAT    100            3     1077         49   4.9  2008 0.0208
##  7 STAT    101            3     1077         59   5.9  2008 0.0250
##  8 STAT    201            3     1077        284  28.4  2008 0.121
##  9 STAT    203            3     1077        164  16.4  2008 0.0696
## 10 STAT    270            3     1077        185  18.5  2008 0.0785
## # ... with 458 more rows
```

Summary of split-apply-combine

- Base R:

```
sp.stat <- split(stat,list(stat$year,stat$Subject))
sp.stat <- lapply(sp.stat,transform,FTEproportion = FTEs/sum(FTEs))
stat <- unsplit(sp.stat,list(stat$year,stat$Subject))
```

- dplyr

```
stat %>% group_by(year,Subject) %>%
  mutate(FTEproportion = FTEs/sum(FTEs)) %>%
  ungroup() -> stat
save(stat,file="statEnrol.RData")
```

Split-apply-combine with `summarise()`

- In the apply step, we may wish to calculate some sort of summary, rather than a transformation of a variable.
- For example, suppose we want to calculate total FTEs by year and subject, and return a data frame

```
stat %>% group_by(year,Subject) %>%
  summarise(totalFTEs = sum(FTEs)) %>%
  ungroup() -> totals
head(totals,n=4)
```

```
## # A tibble: 4 x 3
##    year Subject totalFTEs
##   <dbl> <chr>       <dbl>
## 1  2008 ACMA         20.4
## 2  2008 STAT        236.
## 3  2009 ACMA         18.6
## 4  2009 STAT        213.
```

Split-apply-combine with `lapply()`

- Compare to base R

```
tem <- split(stat,list(stat$year,stat$Subject))
tem <- lapply(tem,function(x) sum(x$FTEs))
tem[1:4]
```

```
## $`2008.ACMA`
## [1] 20.36667
##
## $`2009.ACMA`
## [1] 18.63333
##
## $`2010.ACMA`
## [1] 23.06667
##
## $`2011.ACMA`
## [1] 24.03333
```

- Then would have to write code to coerce output to a data frame.