Lecture 12: Data carpentry with `tidyverse`

STAT598z: Intro. to computing for statistics

Vinayak Rao

Department of Statistics, Purdue University

In [ ]:
```
options(repr.plot.width=5, repr.plot.height=3)
```

We will use a dataset of movies scraped off IMDB: https://www.kaggle.com/deepmatrix/imdb-5000-movie-dataset (https://www.kaggle.com/deepmatrix/imdb-5000-movie-dataset)

- Available from the class website

In [ ]:
```
movies_orig <- read.csv('./Data/movie_metadata.csv')
movies      <- movies_orig
# Can view this in RStudio using View(movies)
movies[1,]
```

In [ ]:
```
unique(movies$director_name)
```

In [ ]:
```
unique(movies$director_name[movies$imdb_score>8.5])
```

In [ ]:
```
(movies$movie_title[movies$imdb_score>9])
```

In [ ]:
```
library('tidyverse')
movies <- as_tibble(movies)
```

Most functions that works with dataframes works with tibbles

- functions in `tidyverse` require tibbles
- additionally, tibbles have some nice conveniences

In [ ]:
```
my_rnd <- tibble(x=rnorm(10), y = x+1, z = x>0)
print(my_rnd) # tibbles also print a bit more nicely
```

The 'pipe' operator %>%

`tidyuniverse` gets this from package `purrr`

- `magrittr` offers additional functionality

A side point on infix functions

%func_name% is syntax for infix (rather than prefix) functions:

In [ ]:
```
'%plus%' <- function(x,y) x+y

1 %plus% 2; '%plus%'(3,4)
```

%>% pipes output of first function to first argument of the second

Can give more readable code. E.g. consider

In [ ]:
```
range(
      movies$actor_1_facebook_likes[
        order(
              movies$imdb_score, decreasing = T
             )
       ][1:10]
     )

# range(movies$actor_1_facebook_likes[
#   order(movies$imdb_score, decreasing = T)][1:10])
```

Have to parse code from inside to outside.

In [ ]:
```
movies$imdb_score %>%
  order(decreasing = T) %>%
  movies$actor_1_facebook_likes[.] %>%  #What are the arguments to '['?
  .[1:10] %>%
  range
```

By default, output of function to left of %>% is the first argument of the function to the right

Use `.` as placeholder if argument you are piping to is not the first

In [ ]:
```
4 %>% log(2) # log(4,2)
```

In [ ]:
```
4 %>% log(2,.) # log(2,4)
```

Can pipe to multiple arguments

In [ ]:
```
2 %>% log(.+6,.) # log(8,2)
```

Pipes in pipes are possible (but be careful)

In [ ]:
```
2 %>% log(.+6 %>% .^2 %>% print,.); log(38,2)
```

`tiyverse` gets %>% from the `purrr` package

The `magrittr` package provides more such functions.

E.g. the T-pipe %T>% passes the LHS onwards

- useful for functions like plot where output isn't important

In [ ]:
```
library(magrittr); rnorm(100) %T>% hist  %>% mean
```

http://www.fromthebottomoftheheap.net/2015/06/03/my-aversion-to-pipes/^ (http://www.fromthebottomoftheheap.net/2015/06/03/my-aversion-to-pipes/^)

https://cran.r-project.org/web/packages/magrittr/vignettes/magrittr.html (https://cran.r-project.org/web/packages/magrittr/vignettes/magrittr.html)

https://www.r-statistics.com/2014/08/simpler-r-coding-with-pipes-the-present-and-future-of-the-magrittr-package/ (https://www.r-statistics.com/2014/08/simpler-r-coding-with-pipes-the-present-and-future-of-the-magrittr-package/)

Our next package from `tidyverse` is `dplyr`

- `filter`: pick observations by values (rows)
- `arrange`: reorder rows
- `select`: pick variables (columns) by their names
- `mutate`: create new variables from existing variables
- `summarise`: summarise many values

The scope of each is determined by `group_by`

For a more thorough overview, look at *R for Data Science* (http://r4ds.had.co.nz/transform.html#datatransformation (http://r4ds.had.co.nz/transform.html#datatransformation))

The filter `filter()` function

Select observations/rows based on value

Cleaner alternative to indexing with logicals and `which`

In [ ]:
```
deniro <- filter(movies,actor_1_name == "Robert De Niro")
deniro[,c('imdb_score','movie_title')]
```

In [ ]:
```
deniro_good <- movies %>%
                filter(actor_1_name == "Robert De Niro",
                       imdb_score > 8)
deniro_good[,c('imdb_score','movie_title')]
```

Multiple argument are equivalent to logical AND (&):

```
deniro_good <- filter(movies,actor_1_name == "Robert De Niro" & imdb_score > 7)
```

Logical or's must be written using |

In [ ]:
```
dnr_pcn<-movies %>%
         filter((actor_1_name=="Robert De Niro") |
                              (actor_1_name=="Al Pacino"),
                           imdb_score > 7)
dnr_pcn[,c('actor_1_name','imdb_score','movie_title')]
```

The `select()` function

Unlike `filter()`, `select()` picks columns of a tibble

In [ ]:
```
select(deniro_good, movie_title, imdb_score)
```

In [ ]:
```
select(deniro_good, director_name:actor_2_name)[1:10,]
```

Can also use `-` to eliminate columns:

In [ ]:
```
select(deniro_good, -(director_name:actor_2_name))
```

Also includes convenience functions like `contains("actor")` and `num_range("var",1:3)`

The `arrange()` function

Orders rows in increasing order of any chosen column

- Additional columns can be provided to break ties
- `desc()` can be used to sort in decreasing order

Missing values always go at the end

In [ ]:
```
movies %>% arrange(desc(imdb_score),desc(gross)) %>%
   select(movie_title, imdb_score, gross) %>% .[1:10,]
```

In [ ]:
```
arrange(movies, imdb_score, gross) %>%
  select(movie_title, imdb_score, gross) %>% .[1:10,]
```

In [ ]:
```
tmp <- arrange(movies, desc(imdb_score), desc(gross))
```

The `mutate()` function

Creates new columns at the end of current data.frame

In [ ]:
```
movies %>% filter(country== "USA") %>%
     select(movie_title, imdb_score, gross, budget) %>%
     mutate(succ = gross/budget) %>%
     arrange(desc(succ)) %>% .[1:20,]
```

`mutate` can refer to functions just created

In [ ]:
```
movies %>% filter(country == "USA") %>%
  select(movie_title, imdb_score, gross, budget) %>%
  mutate(succ = gross-budget, perc= 100*succ/budget) %>%
  distinct() %>% arrange((succ))
```

`distinct()` is a useful function to remove repeated rows

- can provide column names as arguments for partial repetitions

`transmute()` is useful if we only care about the new column

`summarise()` and `group_by()`

Summarise collapses a dataframe to a single row:

In [ ]:
```
summarise(movies, score = mean(imdb_score))
```

Becomes very powerful in conjunction with `group_by()`

In [ ]:
```
top_dir <- movies %>% group_by(director_name) %>%
        summarise(score = mean(imdb_score)) %>%
        arrange(desc(score))
top_dir[1:15,]
```

`n()` is a convenient function to get number of elements

In [ ]:
```
top_dir <- movies %>% group_by(director_name) %>%
        summarise(count=n(), score = mean(imdb_score)) %>%
        arrange(desc(score)) %>%
        filter(count>=5)
top_dir
```

In [ ]:
```
yr_scr <- movies %>% group_by(title_year) %>%
         summarise(count=n(), score = median(imdb_score),
                               ymin = quantile(imdb_score,.1),
                               ymax=quantile(imdb_score,.9)) %>%
         arrange(desc(score)) %>% filter(count>=5)
yr_scr
```

In [ ]:
```
ggplot(yr_scr , aes(x=title_year, y = score)) +
    geom_line() +
    geom_errorbar(aes(ymin=ymin,ymax=ymax))
```

Can have nested groupings (can revert with `ungroup()`)

In [ ]:
```
act_dir<-movies %>% group_by(actor_1_name,director_name) %>%
        distinct(movie_title, .keep_all = T) %>%
        summarise(num = n(), scr = mean(gross-budget),
                  ttl = paste(movie_title, collapse=";")) %>%
        arrange(desc(scr)) %>% filter(num>2)
act_dir[1:20,]
```

Let's try something more complicated:

- Can we analyse scores/earnings across genres?

Things are actually a bit more complicated:

In [ ]:
```
levels(movies$genres)
#movies %>% select(movie_title, genres) %>% .[1:10,]
```

In [ ]:
```
gnr_type <- as.character(levels(movies_orig$genres)) %>%
       strsplit('\\|') %>% #will see regular expressions later
       unlist %>% unique
gnr_type
```

In [ ]:
```
movies <- movies_orig
movies[,gnr_type] <- F
movies$genres <- as.character(movies$genres)
movies[29:54]
```

In [ ]:
```
for(ii in 1:nrow(movies)) { # Will look at better approaches
  movies[ii,gnr_type] <-
   gnr_type %in% strsplit(movies$genres[ii],"\\|")[[1]]
}

colnames(movies)[38] <- "Sci_fi"
colnames(movies)[51] <- "Reality_TV"
colnames(movies)[53] <- "Film_Noir"
colnames(movies)[54] <- "Game_Show"
gnr_type <- colnames(movies)[29:54]
movies[1:10,29:54]
```

In [ ]:
```
#lm(imdb_score ~ Action + Adventure, movies  )
rslt <- lm(paste("gross ~",
            (paste(gnr_type,collapse = '+'))), movies)
rslt
```

In [ ]:
```
movies$ntile <- ntile(movies$imdb_score,10)
movies %>% select(movie_title, ntile) %>% .[1:10,]
```

`summarise_each` let's one summarize multiple columns easily

In [ ]:
```
gnr_frac <- movies %>% group_by(ntile) %>%
                       select(Action:Game_Show)%>%
                       summarise_each(funs(mean, sum))
gnr_frac
```

In [ ]:
```
library('RColorBrewer')
gnr_frac %>%
  gather('Genre','Count',Action:Game_Show) %>% ggplot() +
  geom_line(aes(x=ntile,y=Count,color=Genre,linetype=Genre),
             size=1) +
  scale_linetype_manual(values=c(rep("solid", 12),rep("dashed",11),
                                 rep("twodash",3))) +
  scale_color_manual(values=c(brewer.pal(12,"Set3"),
            brewer.pal(11,"Set3"), brewer.pal(3,"Set3"))) +
  scale_y_log10()
```

`mutate_each` allows you to transform multiple columns

In [ ]:
```
gnr_frac %>% mutate_each(funs(./sum(.)),Action:Game_Show)
```

Generating tidy data

The `gather' function, allows you to combine multiple columns into 2 coloumns.

- turns wide data into tall data

Tall data is useful for e.g. `ggplot`

In [ ]:
```
state_info <- as_tibble(state.x77)
(state_info[1:10,])
```

In [ ]:
```
state_info %>%
  gather(Illiteracy:`HS Grad`, key='InfoType',
         value='InfoValue')%>%
  ggplot +
   geom_smooth(aes(x=Income, y=InfoValue,color=InfoType))
```

In [ ]:
```
movies %>% filter(country=="USA") %>%
     select(title_year, budget, gross)  %>%
    gather(budget,gross, key = 'type', value='amt') %>%
      ggplot + geom_smooth(aes(x=title_year,y=log10(amt),
                               group=type, color=type)) +
      facet_wrap(~type)
```

In [ ]:
```
movies %>% filter(country=="USA") %>%
    select(title_year, budget, Action:Musical)  %>%
    gather(Action:Musical, key = 'type', value='amt') %>%
    filter(amt==TRUE) %>%
      ggplot + geom_smooth(aes(x=title_year,y=log10(budget),
                               group=type, color=type),se=F)
```

`spread()` does the opposite

- turns a tall data.frame into a wide one

Wide data is useful for e.g. `lm`

In [ ]:
```
spread(state_tall, key = InfoType, value=InfoValue)
```

In [ ]:
```
stdnt <- tibble(
  name      = rep(c("Alice", "Bob"), each=4),
  year      = c(2015, 2015, 2016, 2016, 2015, 2015, 2016, 2016),
  semester  = c("Spring","Fall","Spring","Fall", "Spring","Fall","Spring","Fall
"),
  gpa = c(3.2, 3.9, 3.1, 3.6, 3.1, 3.9, 3.3, 3.3)
  )
stdnt
```

In [ ]:
```
stdnt %>% spread(key=semester, value=gpa)
```

What if there are missing/extra values?

`melt()` and `dcast()` from package `reshape` are slightly more powerful

- however, `tidyr` with `splyr` should meet all your needs

Another useful pair of functions in `separate()` and `unite()`

In [ ]:
```
tmp <- movies %>% separate(director_name,c("First","Last"),
                           sep=" ")
```

Can control what to do with missing/extra elements:

tmp <- movies %>% separate(director_name,c("First","Last"), sep=" ", extra="merge", fill="left")

Regular expressions will allow more expressivity

`unite()` does the opposite

In [ ]:
```
tmp <- unite(stdnt, yr_sm, year, semester)
```

In [ ]:
```
spread(tmp, key=yr_sm, value=gpa)
```