Lecture 14: Data carpentry with `tidyverse`

STAT598z: Intro. to computing for statistics

Vinayak Rao

Department of Statistics, Purdue University

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('~/RSRCH/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)
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;
```

%>% 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[.] %>%
  .[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)
```

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

Can pipe to multiple arguments

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

Pipes in pipes are possible (but be careful)

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

`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
```

Our next package from `tidyverse` is `dplyr`

- `filter`: select 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 <- filter(movies,actor_1_name == "Robert De Niro",
                      imdb_score > 7)
deniro_good[,c('imdb_score','movie_title')]
```

Multiple argument are equivalent to logical AND (&):

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

Logical or's must be written using |

In [ ]:
```
dnr_pcn<-filter(movies,(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 [ ]:
```
tmp <- arrange(movies, imdb_score)
```

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

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

The `mutate()` function

Creates new columns at the end of current data.frame

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

Why are some movies making such horrific losses?

`mutate` can refer to functions just created

In [ ]:
```
movies %>%
  select(movie_title, imdb_score, gross, budget) %>%
  mutate(succ = gross-budget, perc= 100*succ/budget) %>%
  distinct %>% arrange(desc(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(score) %>% filter(count>=5)
```

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(imdb_score),
                  ttl = paste(movie_title, collapse=";")) %>%
        arrange(desc(scr)) %>% filter(num>2) %>%
        select(actor_1_name,director_name,num, scr,ttl)
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)
```

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

In [ ]:
```
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)[54] <- "Game_Show"
colnames(movies)[53] <- "Film_Noir"
colnames(movies)[51] <- "Reality_TV"
gnr_type <- colnames(movies)[29:54]
```

In [ ]:
```
rslt <- lm(paste("imdb_score ~", (paste(gnr_type,collapse = '+'))), m
ovies)
rslt
```

In [ ]:
```
movies$ntile <- ntile(movies$imdb_score,10)
```

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