

APPLIED R FOR SOCIAL SCIENTISTS

J. Alexander Branham

April 2016

Available on github:

<https://github.com/jabranham/applied-r-for-social-scientists>

THIS CLASS

- Assumption of *some* previous exposure to R
- We're not explaining assignment, packages, calling functions, etc
- Will *not* be covering statistics
- Alex will start with some common data tasks: loading, variable creation, merging, etc
- Daniel will take the second part to talk about tables and visualization

GETTING THE DATA

- `source()` runs a file through R
- This one checks if you have the data already and tries to download it if not
- The dataset we're using is the General Social Survey spanning 1972-2014

```
source("check-gss-and-maybe-download.R")
```

```
## [1] "GSS file exists!"
```

- R can read almost any data
- Here are some of the most common types:

package	function	file formats
foreign	read.*	dta, spss, etc
haven	read_*	dta (13+) files and others
readr	read_csv	csv files

- We have stata data (*.dta)
- `convert.factors = FALSE` ensures that R doesn't convert the values to the labels that stata uses

```
GSS <- foreign::read.dta("./data/GSS7214_R5.DTA",  
                        convert.factors = FALSE)
```


DATA MANAGEMENT

- Let's get a sense of the data we're working with
- Do you want more, less, or about the same spending? (education and social security)

```
with(GSS, table(nateduc, natsoc, exclude=NULL))
```

##		natsoc			
##	nateduc	1	2	3	<NA>
##	1	8540	4335	699	7939
##	2	2062	2288	296	5005
##	3	353	457	210	1317
##	<NA>	11841	7408	1282	5567

- `cor` gives us (by default) the pearson's r between two variables
- Without setting `use`, R tries to use the whole data, some of which are missing and thus results in `NA`

```
cor(GSS$nateduc,  
    GSS$natsoc,  
    use = "complete.obs")
```

```
## [1] 0.1965368
```

- Let's make an indicator variable for whether a respondent is black or not
- Here's the **race** variable
- I also like to make sure that I'm not going to overwrite an existing variable

```
with(GSS, table(race))
```

```
## race
```

```
##      1      2      3
```

```
## 48240  8312  3047
```

```
with(GSS, table(black))
```

```
## Error in table(black): object 'black' not found
```

- Using `ifelse` to create a variable conditional on other var's values

```
GSS$black <- ifelse(GSS$race == 2, TRUE, FALSE)
with(GSS, table(black))
```

```
## black
## FALSE  TRUE
## 51287  8312
```

- Now let's check to see if that correlation is different for black people
- Note how ugly this looks!

```
with(GSS, cor(nateduc[black == TRUE],  
              natsoc[black == TRUE],  
              use="complete.obs"))
```

```
## [1] 0.1772446
```

- `dplyr` is an R package that makes data management *much* easier
- Different functions for data munging:
- `filter()`, `select()`, `mutate()`
- It introduces the pipe operator `%>%` to the language
- Functions for merging data
- `*_join`: full, inner, left, right
- `group_by`, which lets us perform operations on groups of the data
- Because I'll use `tidyr` later and it gets angry if you load it after `dplyr`, I'm loading it now

```
library(tidyr)  
suppressPackageStartupMessages(library(dplyr))
```

- The pipe (`%>%`) “pipes” the output of the last thing into the first argument of the next thing
- `summarize` (or `summarise`) from `dplyr` returns a `data.frame`

```
with(filter(GSS, black == TRUE),  
      cor(nateduc, natsoc,  
          use = "complete.obs"))
```

```
## [1] 0.1772446
```



```
GSS %>%  
  filter(black == TRUE) %>%  
  summarize(mycor =  
    cor(nateduc, natsoc,  
      use = "complete.obs"))
```

```
##          mycor  
## 1 0.1772446
```

- 1972 doesn't have any observations we're interested in (our spending variables weren't asked), so let's drop it
- Again, we can use `filter`, but this time we assign the result back to `GSS`:

```
GSS <- GSS %>%  
  filter(year != 1972)
```

- Variables with categories can be represented as factors in R
- If you want R to think they're ordered, you can use **ordered** TRUE= as an argument

```
table(GSS$sex)
```

```
##
```

```
##      1      2
```

```
## 25479 32507
```

```
GSS <- GSS %>%  
  mutate(sex = factor(sex,  
                       levels = c(1, 2),  
                       labels = c("M", "F")))  
  
with(GSS, table(sex))  
  
## sex  
##      M      F  
## 25479 32507
```

- `dplyr` provides `group_by`
- Lets us perform operations to grouped data

```
thecors <- GSS %>%  
  group_by(sex, black) %>%  
  summarize(thecor = cor(nateduc, natsoc,  
                        use = "complete.obs"),  
            n = n())
```

```
print(thecors)
```

```
## Source: local data frame [4 x 4]
```

```
## Groups: sex [?]
```

```
##
```

```
##      sex black    thecor      n
```

```
## (fctr) (lgl)    (dbl) (int)
```

```
## 1      M FALSE 0.1918454 22446
```

```
## 2      M  TRUE 0.1674413  3033
```

```
## 3      F FALSE 0.1786193 27489
```

```
## 4      F  TRUE 0.1820090  5018
```

- Maybe we're interested in preferences by year?

```
gss_yearly <- GSS %>%  
  group_by(year) %>%  
  summarize(educ = mean(nateduc,  
                        na.rm = TRUE),  
            soc = mean(natsoc,  
                        na.rm = TRUE))
```

```
head(gss_yearly)
```

```
## Source: local data frame [6 x 3]
```

```
##
```

```
##   year      educ   soc
```

```
## (int)    (dbl) (dbl)
```

```
## 1  1973 1.582287  NaN
```

```
## 2  1974 1.562059  NaN
```

```
## 3  1975 1.604930  NaN
```

```
## 4  1976 1.579020  NaN
```

```
## 5  1977 1.605854  NaN
```

```
## 6  1978 1.576766  NaN
```


- Means are nice, but there are other ways to summarize data
- What if we want to look at the proportion of people who support more spending minus the proportion who support less?

```
netsupport <- function(thedata){  
  prop_more <- mean(thedata == 1, na.rm = TRUE)  
  prop_less <- mean(thedata == 3, na.rm = TRUE)  
  prop_more - prop_less  
}
```

```
GSS %>%
```

```
  group_by(year) %>%
```

```
  summarize(support_educ = netsupport(nateduc),
```

```
            support_soc = netsupport(natsoc))
```

```
## Source: local data frame [29 x 3]
```

```
##
```

```
##      year support_educ support_soc
```

```
##    (int)         (dbl)         (dbl)
```

```
## 1   1973      0.4177127          NA
```

```
## 2   1974      0.4379408          NA
```

```
## 3   1975      0.3950704          NA
```

```
## 4   1976      0.4209800          NA
```

```
## 5   1977      0.3941457          NA
```

- The `ggplot2` package provides the `economics` data.frame that has US economic data starting in July 1967
- `?economics` gives more info

```
library(ggplot2)
head(economics, 3)
```

```
## Source: local data frame [3 x 6]
##
##      date    pce    pop psavert uempmed unemploy
##   (date) (dbl) (int)  (dbl)  (dbl)   (int)
## 1 1967-07-01 507.4 198712   12.5    4.5   2944
## 2 1967-08-01 510.5 198911   12.5    4.7   2945
## 3 1967-09-01 516.3 199113   11.7    4.6   2958
```

- Let's make an unemployment rate by `unemploy*pop`

```
economics <- economics %>%  
  mutate(unemp_rate = unemploy / pop)
```

- Note `mutate` is from `dplyr`, this is base R:

```
economics$unemp_rate <- economics$unemploy / economics$pop
```

- The `economics` data is monthly and our GSS data is yearly, so we need to aggregate

```
economics_yearly <- economics %>%  
  mutate(year = format(date, "%Y")) %>%  
  group_by(year) %>%  
  summarize(unemp = mean(unemp_rate))
```

- Let's see what our data looks like now!

```
head(economics_yearly)
```

```
## Source: local data frame [6 x 2]
```

```
##
```

```
##   year      unemp
```

```
##   (chr)      (dbl)
```

```
## 1  1967 0.01512179
```

```
## 2  1968 0.01394202
```

```
## 3  1969 0.01396464
```

```
## 4  1970 0.02012547
```

```
## 5  1971 0.02418970
```

```
## 6  1972 0.02323808
```

- Now we have two `data.frame` objects — `gss_yearly` and `economics_yearly` — that we want to join together
- `dplyr` provides a really easy way of doing this
- The jargon comes from SQL, a programming language used to store data
- What you probably call a “merge” `dplyr` calls a “join”
- `*_join` where `*` is either `full`, `inner`, `left`, or `right`
- We’ll use `left_join` since the economics data contains years that aren’t in the GSS

```
gss_yearly <- left_join(gss_yearly,  
                        economics_yearly,  
                        by = "year")
```

```
## Error in eval(expr, envir, enclos): cannot join on columns 'year' x 'year': C
```


- Error: cannot join on columns 'year' x 'year': Can't join on 'year' x 'year' because of incompatible types (character / integer)

- The error on the last slide indicates that the **year** variable in the two datasets is different
- Let's verify that:

```
class(gss_yearly$year)
```

```
## [1] "integer"
```

```
class(economics_yearly$year)
```

```
## [1] "character"
```

- Solution: change `economics_yearly$year` to an integer

```
economics_yearly$year <- as.integer(economics_yearly$year)
```

```
gss_yearly <- left_join(gss_yearly,  
                        economics_yearly,  
                        by="year")
```

```
head(gss_yearly)
```

```
## Source: local data frame [6 x 4]
##
##   year      educ   soc      unemp
##   (int)    (dbl) (dbl)    (dbl)
## 1  1973 1.582287   NaN 0.02057710
## 2  1974 1.562059   NaN 0.02418823
## 3  1975 1.604930   NaN 0.03677624
## 4  1976 1.579020   NaN 0.03393594
## 5  1977 1.605854   NaN 0.03164388
## 6  1978 1.576766   NaN 0.02780555
```

- Maybe you want to save this new data so you don't have to re-run the merging whenever you want to

package	function	result
readr	write_csv	csv file
utils	write.csv	csv file
base	save	Rdata file
xlsx	write.xlsx	excel file

- R can also write to stata/SPSS/SAS files through **foreign** or **haven**

- Let's save a csv file
- If the **data/** subfolder doesn't exist, this will produce an error
- The script that we ran at the beginning created this if it didn't already exist

```
readr::write_csv(gss_yearly, "data/gss-yearly-data.csv")
```

TIDYING DATA

- Sometimes the data you get aren't *tidy*
- Tidy data are data where each row is an observation, each column a variable, and each cell a value
- Most of the strategies I showed you above assume that you're dealing with tidy data
- Remember I loaded `tidyr` earlier, so there's no need to call `library` again


```
messy1 <- data_frame(  
  country = c("Afghanistan", "Albania", "Algeria"),  
  "2007" = c(43.82, 76.42, 72.30),  
  "2002" = c(42.13, 75.65, 70.99))
```

```
print(messy1)
```

```
## Source: local data frame [3 x 3]
```

```
##
```

```
##      country  2007  2002
```

```
##      (chr) (dbl) (dbl)
```

```
## 1 Afghanistan 43.82 42.13
```

```
## 2      Albania 76.42 75.65
```

```
## 3      Algeria 72.30 70.99
```

- `gather` can also turn wide to long

```
gather(messy1, "year", "life_expect", 2:3)
```

```
## Source: local data frame [6 x 3]
```

```
##
```

```
##      country  year life_expect
```

```
##      (chr) (chr)      (dbl)
```

```
## 1 Afghanistan 2007      43.82
```

```
## 2     Albania 2007      76.42
```

```
## 3     Algeria 2007      72.30
```

```
## 4 Afghanistan 2002      42.13
```

```
## 5     Albania 2002      75.65
```

```
## 6     Algeria 2002      70.99
```

```
messy2 <- data.frame(  
  country = c(rep("Afghanistan", 4), rep("Albania", 4), rep("Algeria", 4)),  
  year = c(rep(2002, 2), rep(2007, 2)),  
  variable = c("life_expect", "pop"),  
  value = c(42.12, 25268405, 43.82, 31889923,  
            75.65, 3508512, 76.42, 3600523,  
            70.99, 31287142, 72.30, 33333216)  
)
```

```
head(messy2)
```

```
##      country year  variable      value
## 1 Afghanistan 2002 life_expect  42.12
## 2 Afghanistan 2002      pop 25268405.00
## 3 Afghanistan 2007 life_expect  43.82
## 4 Afghanistan 2007      pop 31889923.00
## 5      Albania 2002 life_expect  75.65
## 6      Albania 2002      pop  3508512.00
```

- `spread` can also turn long to wide

```
spread(messy2, key = variable, value)
```

```
##      country year life_expect      pop
## 1 Afghanistan 2002      42.12 25268405
## 2 Afghanistan 2007      43.82 31889923
## 3     Albania 2002      75.65  3508512
## 4     Albania 2007      76.42  3600523
## 5     Algeria 2002      70.99 31287142
## 6     Algeria 2007      72.30 33333216
```

- If you have two variables in one column, use **separate**
- For example, a rate of # of people with a trait / total population in each country
- One variable across two columns? use **unite**
- one column for century (19, 20) and another for year (00... 09)

ITERATION: LOOPS AND APPLY

- DRY (*Don't Repeat Yourself*) is an acronym from computer science
- Repeating yourself makes your code harder to deal with:
- Intent is less clear
- Harder to spot bugs

- For loops get lots of hate online because people think they're slow (they aren't)
- They can be hard to read, though

```
thedata <- data_frame(  
  one = rnorm(100), two = rnorm(100),  
  three = rnorm(100), four = rnorm(100)  
)
```

```
output <- list()
output[[1]] <- median(thedata$one)
output[[2]] <- median(thedata$two)
output[[3]] <- median(thedata$three)
output[[4]] <- median(thedata$four)
print(output); rm(output)
```

```
## [[1]]
## [1] -0.04354002
##
## [[2]]
## [1] -0.007102403
##
## [[3]]
```

```
output <- list()
for (i in 1:4) {
  output[[i]] <- median(thedata[[i]])
}
print(output); rm(output)
```

```
## [[1]]
## [1] -0.04354002
##
## [[2]]
## [1] -0.007102403
##
## [[3]]
## [1] 0.1239376
```

- Of course, we oftentimes need to perform an operation across many columns
- This is where the `map` family (from `purrr`) steps in:

```
suppressPackageStartupMessages(library(purrr))  
map(thedata, median)
```

```
## $one  
## [1] -0.04354002  
##  
## $two  
## [1] -0.007102403  
##  
## $three  
## [1] 0.1239376  
##
```

```
map_dbl(thedata, median)
```

```
##           one           two           three           four
## -0.043540022 -0.007102403  0.123937598 -0.046944064
```

NESTED DATA

- Some data is nested in a hierarchical way
- the `gapminder` data are a good example¹

```
library(gapminder); library(ggplot2)
head(gapminder, 3)
```

```
## Source: local data frame [3 x 6]
```

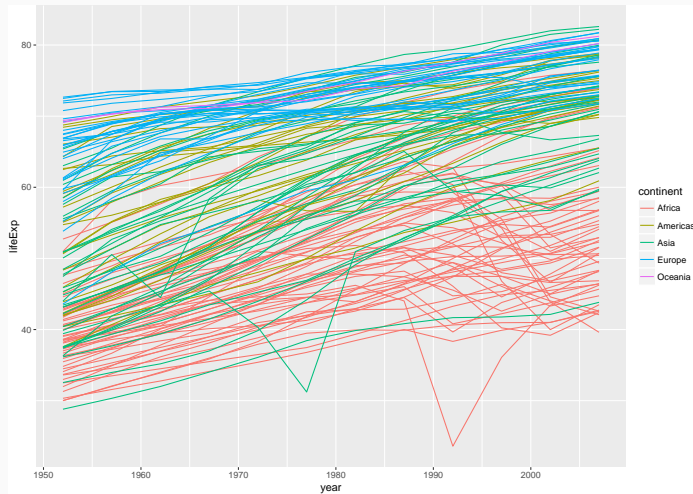
```
##
```

```
##      country continent  year lifeExp      pop gdpPercap
##      (fctr)   (fctr) (int)  (dbl)    (int)    (dbl)
## 1 Afghanistan      Asia  1952  28.801  8425333  779.4453
## 2 Afghanistan      Asia  1957  30.332  9240934  820.8530
## 3 Afghanistan      Asia  1962  31.997 10267083  853.1007
```

¹This example taken from the blog post


```
ggplot(gapminder, aes(x = year, y = lifeExp,  
                      color = continent, by = country)) +  
  geom_line()
```

LIFE EXPECTANCY OVER TIME



```
by_country <- gapminder %>%  
  group_by(continent, country) %>%  
  nest()
```

- Now we have a data frame with one row per group and a column where each cell is itself a whole data frame

```
head(by_country,3)
```

```
## Source: local data frame [3 x 3]
##
##   continent      country      data
##   (fctr)        (fctr)      (chr)
## 1      Asia Afghanistan <tbl_df [12,4]>
## 2    Europe    Albania <tbl_df [12,4]>
## 3    Africa    Algeria <tbl_df [12,4]>
```

- So for example the first element of the **data** column contains the whole data frame for Afghanistan

```
by_country$data[[1]]
```

```
## Source: local data frame [12 x 4]
```

```
##
```

```
##      year lifeExp      pop gdpPercap
```

```
##      (int)  (dbl)    (int)      (dbl)
```

```
## 1   1952  28.801  8425333  779.4453
```

```
## 2   1957  30.332  9240934  820.8530
```

```
## 3   1962  31.997 10267083  853.1007
```

```
## 4   1967  34.020 11537966  836.1971
```

```
## 5   1972  36.088 13079460  739.9811
```

```
## 6   1977  38.422 14222272  726.1124
```

- You can create a linear model for each country then:

```
by_country <- by_country %>%  
  mutate(model = map(data,  
    ~ lm(lifeExp ~ year, data = .)))
```

```
head(by_country, 3)
```

```
## Source: local data frame [3 x 4]
```

```
##
```

```
##   continent      country      data      model  
##   (fctr)      (fctr)      (chr)      (chr)  
## 1      Asia Afghanistan <tbl_df [12,4]> <S3:lm>  
## 2     Europe      Albania <tbl_df [12,4]> <S3:lm>  
## 3     Africa      Algeria <tbl_df [12,4]> <S3:lm>
```

- Here we can extract the fitted values and plot a line of the fitted values
- By continent, country

```
by_country %>% unnest(model %>% map(broom::augment)) %>%  
  select(continent, country, year, .fitted) %>%  
  ggplot(aes(x = year, y = .fitted,  
             by = country, color = continent)) +  
  geom_line()
```