

Stat 341 Lecture 3

Brad McNeney

2019-01-17

Subsetting with `dplyr`

Control Flow

Reading from and writing to files

Subsetting with dplyr

Subsetting with dplyr

- ▶ Subsetting of data frames in the tidyverse is done with the `filter()` and `select()` functions from `dplyr`.
- ▶ We will discuss filtering and selecting in much more detail later in the course.

```
library(gapminder) # contains the data
library(dplyr)
```

- ▶ Take a look at the top and bottom few lines of raw data.

```
head(gapminder)
```

```
## # A tibble: 6 x 6
##   country      continent  year lifeExp      pop gdpPercap
##   <fct>        <fct>    <int>   <dbl>   <int>    <dbl>
## 1 Afghanistan Asia      1952    28.8  8425333    779.
## 2 Afghanistan Asia      1957    30.3  9240934    821.
## 3 Afghanistan Asia      1962    32.0 10267083    853.
## 4 Afghanistan Asia      1967    34.0 11537966    836.
## 5 Afghanistan Asia      1972    36.1 13079460    740.
## 6 Afghanistan Asia      1977    38.4 14880372    786.
```

```
tail(gapminder)
```

```
## # A tibble: 6 x 6
```

##	country	continent	year	lifeExp	pop	gdpPercap
##	<fct>	<fct>	<int>	<dbl>	<int>	<dbl>
## 1	Zimbabwe	Africa	1982	60.4	7636524	789.
## 2	Zimbabwe	Africa	1987	62.4	9216418	706.
## 3	Zimbabwe	Africa	1992	60.4	10704340	693.
## 4	Zimbabwe	Africa	1997	46.8	11404948	792.
## 5	Zimbabwe	Africa	2002	40.0	11926563	672.
## 6	Zimbabwe	Africa	2007	43.5	12311143	470.

```
summary(gapminder)
```

```
##           country      continent      year      lifeExp
## Afghanistan: 12 Africa :624 Min. :1952 Min. :23.60
## Albania : 12 Americas:300 1st Qu.:1966 1st Qu.:48.20
## Algeria : 12 Asia :396 Median :1980 Median :60.71
## Angola : 12 Europe :360 Mean :1980 Mean :59.47
## Argentina : 12 Oceania : 24 3rd Qu.:1993 3rd Qu.:70.85
## Australia : 12 Max. :2007 Max. :82.60
## (Other) :1632
##           pop      gdpPercap
## Min. :6.001e+04 Min. : 241.2
## 1st Qu.:2.794e+06 1st Qu.: 1202.1
## Median :7.024e+06 Median : 3531.8
## Mean :2.960e+07 Mean : 7215.3
## 3rd Qu.:1.959e+07 3rd Qu.: 9325.5
## Max. :1.319e+09 Max. :113523.1
##
```

Tibbles

- ▶ Tibbles are an implementation of data frames that is developing as the standard data structure in the tidyverse.
- ▶ See the documentation for the differences between data frames and tibbles: <https://cran.r-project.org/web/packages/tibble/vignettes/tibble.html>
- ▶ Coerce a data frame to a tibble with `as_tibble()` and a tibble to a data frame with `as.data.frame()`.

Select rows with filter()

- Extract the data from 2007:

```
gapminder07 <- filter(gapminder, year == 2007)
head(gapminder07)
```

```
## # A tibble: 6 x 6
##   country      continent  year lifeExp      pop gdpPercap
##   <fct>        <fct>    <int>  <dbl>    <int>    <dbl>
## 1 Afghanistan Asia      2007   43.8  31889923    975.
## 2 Albania     Europe    2007   76.4   3600523   5937.
## 3 Algeria     Africa    2007   72.3  33333216   6223.
## 4 Angola      Africa    2007   42.7  12420476   4797.
## 5 Argentina   Americas  2007   75.3  40301927  12779.
## 6 Australia   Oceania   2007   81.2  20434176  34435.
```

Select columns with `select()`

- ▶ Extract the year, life expectancy and country with `select`.

```
gmReduced <- select(gapminder, year, lifeExp, country)
gmReduced <- select(gapminder, -continent, -pop, -gdpPercap)
```

Chaining with the Forward Pipe

- ▶ Chain together multiple data manipulations with the forward pipe `%>%`.
 - ▶ The forward pipe is from the `magrittr` package, but is available automatically when we load `dplyr`.

```
gapminder %>%  
  filter(year == 2007) %>%  
  select(year, lifeExp, country) %>%  
  head()
```

```
## # A tibble: 6 x 3  
##   year lifeExp country  
##   <int>   <dbl> <fct>  
## 1  2007    43.8 Afghanistan  
## 2  2007    76.4 Albania  
## 3  2007    72.3 Algeria  
## 4  2007    42.7 Angola  
## 5  2007    75.3 Argentina  
## 6  2007    81.2 Australia
```

Control Flow

if and if-else

- ▶ if tests a condition and executes code if the condition is true. Optionally, can couple with an else to specify code to execute when condition is false.

```
if("cat" == "dog") {  
  print("cat is dog")  
} else {  
  print("cat is not dog")  
}
```

```
## [1] "cat is not dog"
```

for loops

Example:

```
n <- 10; nreps <- 100; x <- vector(mode="numeric",length=nreps)
for(i in 1:nreps) {
  # Code you want to repeat nreps times
  x[i] <- mean(rnorm(n))
}
summary(x)
```

```
##      Min.   1st Qu.   Median     Mean   3rd Qu.     Max.
## -0.83562 -0.26567 -0.02293 -0.03405  0.18281  0.77689
```

```
print(i)
```

```
## [1] 100
```

for loop index set

- ▶ Index sets of the form 1:n are most common, but can be almost any atomic vector.

```
ind <- c("cat", "dog", "mouse")
for(i in ind) {
  print(paste("There is a", i, "in my house"))
}
```

```
## [1] "There is a cat in my house"
## [1] "There is a dog in my house"
## [1] "There is a mouse in my house"
```

while loops

- Use a while loop when you want to continue until some logical condition is met.

```
set.seed(1)
# Number of coin tosses until first success (geometric distn)
p <- 0.1; counter <- 0; success <- FALSE
while(!success) {
  success <- as.logical(rbinom(n=1,size=1,prob=p))
  counter <- counter + 1
}
counter
```

```
## [1] 4
```


break

- ▶ break can be used to break out of a for or while loop.

```
for(i in 1:100) {  
  if(i>3) break  
  print(i)  
}
```

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

```
## [1] 2
```

```
## [1] 3
```

Reading from and writing to files

Native format

- ▶ Use `save()` to save R objects to an “R Data” file.
 - ▶ `save.image()` is short-hand to save all objects in the workspace

```
x <- rnorm(100); y <- list(a=1,x=x)
save(x,y,file="test.RData") # Or .rda, or ...
```

- ▶ Load R Data files into the workspace with `load()`.

```
load("test.RData")
file.remove("test.RData")
```

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

Table format files

- ▶ `read.table()` is the main function for reading tabular data from plain-text files.
 - ▶ `read.csv()` and `read.delim()` are basically `read.table()` with defaults for reading comma- and tab- delimited files.
- ▶ `write.table()`, `write.csv()` and `write.delim()` are the analogous functions for writing tabular data

```
write.table(matrix(1:9,3,3),file="test.txt")
test <- read.table("test.txt")
file.remove("test.txt")
```

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

```
test
```

```
##    V1 V2 V3
## 1   1  4  7
## 2   2  5  8
## 3   3  6  9
```

Reading files from a URL

- `load()`, `read.table()`, etc. can read data from a URL.

```
baseURL <- "http://people.stat.sfu.ca/~mcneney/Teaching/Stat341/"
rdURL <- url(paste0(baseURL,"Data/PorschePrice.rda"))
load(rdURL)
head(PorschePrice)
```

```
##   Price Age Mileage
## 1  69.4   3   21.5
## 2  56.9   3   43.0
## 3  49.9   2   19.9
## 4  47.4   4   36.0
## 5  42.9   4   44.0
## 6  36.9   6   49.8
```

```
csvURL <- url(paste0(baseURL,"Data/PorschePrice.csv"))
PorschePrice <- read.csv(csvURL)
```

Reading more complex text files

- ▶ Defaults for `read.table()` are not always what you want.
 - ▶ In particular, the default for reading columns that include text is to coerce to a factor.
 - ▶ Also replaces spaces in column headers with `..`

```
exURL <- url(paste0(baseURL, "Data/Ex1_1_4.txt"))
ex <- read.table(exURL, header=TRUE, sep="\t")
# same as ex <- read.delim(exURL)
ex
```

```
##   ID Initials Date.of.purchase amount
## 1  3      SEKK      10/23/1995  $5.00
## 2  1      AGKE      08/03/1999 $10.49
## 3  2      SBKE      12/18/2002 $11.00
```

```
str(ex)
```

```
## 'data.frame':    3 obs. of  4 variables:
## $ ID              : int  3 1 2
## $ Initials        : Factor w/ 3 levels "AGKE","SBKE",...: 3 1 2
## $ Date.of.purchase: Factor w/ 3 levels "08/03/1999","10/23/1995",...: 2 1 3
## $ amount          : Factor w/ 3 levels "$10.49","$11.00",...: 3 1 2
```

stringsAsFactors

- ▶ Reading columns that include characters in as factors is controlled by a global option in your R session called `stringsAsFactors`, set to `TRUE` by default.
- ▶ If you want to set to `FALSE` for an R session type `options(stringsAsFactors = FALSE)` into the Console.
- ▶ An alternative is to over-ride the default in the call to `read.table()`:

```
exURL <- url(paste0(baseUrl,"Data/Ex1_1_4.txt"))
ex2 <- read.table(exURL,header=TRUE,sep="\t",
                  stringsAsFactors=FALSE)
str(ex2)
```

```
## 'data.frame':    3 obs. of  4 variables:
## $ ID             : int  3 1 2
## $ Initials       : chr  "SEKK" "AGKE" "SBKE"
## $ Date.of.purchase: chr  "10/23/1995" "08/03/1999" "12/18/2002"
## $ amount         : chr  "$5.00" "$10.49" "$11.00"
```

Post-processing: dates

- `Date.of.purchase` should be coerced to a `Date` object.

```
ex2$Date.of.purchase <-  
  as.Date(ex2$Date.of.purchase, "%m/%d/%Y")  
str(ex2)
```

```
## 'data.frame':    3 obs. of  4 variables:  
## $ ID           : int  3 1 2  
## $ Initials     : chr  "SEKK" "AGKE" "SBKE"  
## $ Date.of.purchase: Date, format: "1995-10-23" "1999-08-03" ...  
## $ amount       : chr  "$5.00" "$10.49" "$11.00"
```

```
diff(ex2$Date.of.purchase)
```

```
## Time differences in days  
## [1] 1380 1233
```


Post-processing: strings

- ▶ Will probably want to remove the \$ in amount and coerce to numeric.
- ▶ Many options for manipulating strings – will discuss in detail later in the course.
- ▶ For now, just mention substr() and paste():

```
maxStringLen <- 10 # allows for amounts up to $999999.99
ex2$amount <- as.numeric(
  substr(ex2$amount, start=2, stop=maxStringLen)
)
str(ex2)
```

```
## 'data.frame':    3 obs. of  4 variables:
## $ ID             : int  3 1 2
## $ Initials       : chr  "SEKK" "AGKE" "SBKE"
## $ Date.of.purchase: Date, format: "1995-10-23" "1999-08-03" ...
## $ amount         : num  5 10.5 11
```

```
paste("$", ex2$amount, sep="") # or paste0("$", ex2$amount)
```

```
## [1] "$5"      "$10.49" "$11"
```

Reading and writing Excel files

- ▶ If you have a working copy of Excel, you can export to .csv format and use `read.csv()` and `write.csv()`.
- ▶ However, there are functions in several R packages for reading directly from an Excel file.
 - ▶ E.G., see the `read_excel()` function from the `readxl` package.
 - ▶ Or, try `read.xls()` from the `gdata` package.

Reading data with readr

- ▶ The readr package provides tidyverse equivalents to the read. functions from base R.
 - ▶ The equivalents are of basically the same name, with the dot replaced by an underscore; e.g., `read_table()` replaces `read.table()`.
- ▶ These equivalents (i) return a tibble and (ii) try to correctly parse strings and dates.
- ▶ See the documentation at <http://readr.tidyverse.org/> for full details.

```
library(readr) # or library(tidyverse)
exURL <- url(paste0(baseURL,"Data/Ex1_1_4.txt"))
ex2 <- read_tsv(exURL,col_names=TRUE) # tab-separated
ex2 # Date of purchase not coerced to Date
```

```
## # A tibble: 3 x 4
##       ID Initials `Date of purchase` amount
##   <int> <chr>      <chr>              <chr>
## 1     3 SEKK      10/23/1995          $5.00
## 2     1 AGKE      08/03/1999         $10.49
## 3     2 SBKE      12/18/2002         $11.00
```