dplyr functions for a single dataset

Jenny Bryan, modified by Roger H. French

29 September, 2018

Contents

3.2.4.1 Where were we?

In the introduction to `plyr`,

- we used two very important verbs and an operator:
  - `filter()` for subsetting data row-wise
  - `select()` for subsetting data variable- or column-wise
  - the pipe operator `%>%`, which feeds the LHS as the first argument
  - to the expression on the RHS

Here we explore other `dplyr` functions,

- especially more verbs,
- for working with a single dataset.

3.2.4.1.1 Load `dplyr` and the Gapminder data

We use an excerpt of the Gapminder data

- and store it as a `tbl_df` object, an enhanced data.frame.

I'll use the pipe operator even here,

- to demonstrate its utility outside of `dplyr`.

```
suppressPackageStartupMessages(library(dplyr))
# gd_url <- "http://tiny.cc/gapminder"
gd_url <- "http://www.stat.ubc.ca/~jenny/notOcto/STAT545A/examples/gapminder/data/gapminderDataFiveYear
gtbl <- gd_url %>% read.delim %>% tbl_df
gtbl %>% glimpse
```

```
## Observations: 1,704
## Variables: 6
## $ country   <fct> Afghanistan, Afghanistan, Afghanistan, Afghanistan, ...
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992...
## $ pop       <dbl> 8425333, 9240934, 10267083, 11537966, 13079460, 1488...
```

```
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia...
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.8...
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 78...
```

3.2.4.2 Use `mutate()` to add new variables

Imagine we wanted to recover each country's GDP.

After all, the Gapminder data has

- a variable for population
- and a variable for GDP per capita.
- Let's multiply them together.

```
gtbl <- gtbl %>%
  mutate(gdp = pop * gdpPercap)
gtbl %>% glimpse
```

```
## Observations: 1,704
## Variables: 7
## $ country   <fct> Afghanistan, Afghanistan, Afghanistan, Afghanistan, ...
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992...
## $ pop       <dbl> 8425333, 9240934, 10267083, 11537966, 13079460, 1488...
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia...
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.8...
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 78...
## $ gdp       <dbl> 6567086330, 7585448670, 8758855797, 9648014150, 9678...
```

Hmmmm ... those GDP numbers are almost uselessly large and abstract.

Consider the advice of Randall Munroe of xkcd:

"One thing that bothers me is large numbers presented without context... 'If I added a zero to this number, would the sentence containing it mean something different to me?'

If the answer is 'no,' maybe the number has no business being in the sentence in the first place."

Maybe it would be more meaningful to consumers of my tables and figures if I reported GDP per capita, *relative to some benchmark country*.

Since Canada is my (Jenny Bryan's) adopted home, I'll go with that.

```
just_canada <- gtbl %>% filter(country == "Canada")
gtbl <- gtbl %>%
  mutate(canada = just_canada$gdpPercap[match(year, just_canada$year)],
         gdpPercapRel = gdpPercap / canada)
gtbl %>%
  select(country, year, gdpPercap, canada, gdpPercapRel)
```

```
## # A tibble: 1,704 x 5
##    country      year gdpPercap canada gdpPercapRel
##    <fct>       <int>     <dbl>  <dbl>        <dbl>
##  1 Afghanistan  1952      779. 11367.       0.0686
##  2 Afghanistan  1957      821. 12490.       0.0657
##  3 Afghanistan  1962      853. 13462.       0.0634
##  4 Afghanistan  1967      836. 16077.       0.0520
##  5 Afghanistan  1972      740. 18971.       0.0390
##  6 Afghanistan  1977      786. 22091.       0.0356
##  7 Afghanistan  1982      978. 22899.       0.0427
```

```
##  8 Afghanistan  1987      852. 26627.       0.0320
##  9 Afghanistan  1992      649. 26343.       0.0246
## 10 Afghanistan  1997      635. 28955.       0.0219
## # ... with 1,694 more rows
```

```
gtbl %>%
  select(gdpPercapRel) %>%
  summary
```

```
##   gdpPercapRel
##  Min.   :0.007236
##  1st Qu.:0.061648
##  Median :0.171521
##  Mean   :0.326659
##  3rd Qu.:0.446564
##  Max.   :9.534690
```

Note that, `mutate()`

- builds new variables sequentially
- so you can reference earlier ones (like `canada`)
- when defining later ones (like `gdpPercapRel`).

(I got a little off topic here using `match()`

- to do table look up,
- but you can figure that out.)

The relative GDP per capita numbers are, in general, well below 1.

We see that most of the countries covered by this dataset

- have substantially lower GDP per capita,
  - relative to Canada,
- across the entire time period.

3.2.4.3 Use `arrange()` to row-order data in a principled way

Imagine you wanted this data ordered

- by year then country,
- as opposed to by country then year.

```
gtbl %>%
  arrange(year, country)
```

```
## # A tibble: 1,704 x 9
##    country  year    pop continent lifeExp gdpPercap     gdp canada
##    <fct>   <int>  <dbl> <fct>       <dbl>     <dbl>   <dbl>  <dbl>
##  1 Afghan~  1952 8.43e6 Asia         28.8      779. 6.57e 9 11367.
##  2 Albania  1952 1.28e6 Europe       55.2     1601. 2.05e 9 11367.
##  3 Algeria  1952 9.28e6 Africa       43.1     2449. 2.27e10 11367.
##  4 Angola   1952 4.23e6 Africa       30.0     3521. 1.49e10 11367.
##  5 Argent~  1952 1.79e7 Americas     62.5     5911. 1.06e11 11367.
##  6 Austra~  1952 8.69e6 Oceania      69.1    10040. 8.73e10 11367.
##  7 Austria  1952 6.93e6 Europe       66.8     6137. 4.25e10 11367.
##  8 Bahrain  1952 1.20e5 Asia         50.9     9867. 1.19e 9 11367.
##  9 Bangla~  1952 4.69e7 Asia         37.5      684. 3.21e10 11367.
## 10 Belgium  1952 8.73e6 Europe       68       8343. 7.28e10 11367.
```

```
## # ... with 1,694 more rows, and 1 more variable: gdpPercapRel <dbl>
```

Or maybe you want just the data from 2007,

- sorted on life expectancy?

```
gtbl %>%
  filter(year == 2007) %>%
  arrange(lifeExp)
```

```
## # A tibble: 142 x 9
##    country  year    pop continent lifeExp gdpPercap     gdp canada
##    <fct>   <int>  <dbl> <fct>       <dbl>     <dbl>   <dbl>  <dbl>
##  1 Swazil~  2007 1.13e6 Africa       39.6     4513. 5.11e 9 36319.
##  2 Mozamb~  2007 2.00e7 Africa       42.1      824. 1.64e10 36319.
##  3 Zambia   2007 1.17e7 Africa       42.4     1271. 1.49e10 36319.
##  4 Sierra~  2007 6.14e6 Africa       42.6      863. 5.30e 9 36319.
##  5 Lesotho  2007 2.01e6 Africa       42.6     1569. 3.16e 9 36319.
##  6 Angola   2007 1.24e7 Africa       42.7     4797. 5.96e10 36319.
##  7 Zimbab~  2007 1.23e7 Africa       43.5      470. 5.78e 9 36319.
##  8 Afghan~  2007 3.19e7 Asia         43.8      975. 3.11e10 36319.
##  9 Centra~  2007 4.37e6 Africa       44.7      706. 3.08e 9 36319.
## 10 Liberia  2007 3.19e6 Africa       45.7      415. 1.32e 9 36319.
## # ... with 132 more rows, and 1 more variable: gdpPercapRel <dbl>
```

Oh, you'd like to sort on life expectancy

- in ___desc___ending order?
- Then use `desc()`.

```
gtbl %>%
  filter(year == 2007) %>%
  arrange(desc(lifeExp))
```

```
## # A tibble: 142 x 9
##    country  year    pop continent lifeExp gdpPercap     gdp canada
##    <fct>   <int>  <dbl> <fct>       <dbl>     <dbl>   <dbl>  <dbl>
##  1 Japan    2007 1.27e8 Asia         82.6    31656. 4.04e12 36319.
##  2 Hong K~  2007 6.98e6 Asia         82.2    39725. 2.77e11 36319.
##  3 Iceland  2007 3.02e5 Europe       81.8    36181. 1.09e10 36319.
##  4 Switze~  2007 7.55e6 Europe       81.7    37506. 2.83e11 36319.
##  5 Austra~  2007 2.04e7 Oceania      81.2    34435. 7.04e11 36319.
##  6 Spain    2007 4.04e7 Europe       80.9    28821. 1.17e12 36319.
##  7 Sweden   2007 9.03e6 Europe       80.9    33860. 3.06e11 36319.
##  8 Israel   2007 6.43e6 Asia         80.7    25523. 1.64e11 36319.
##  9 France   2007 6.11e7 Europe       80.7    30470. 1.86e12 36319.
## 10 Canada   2007 3.34e7 Americas     80.7    36319. 1.21e12 36319.
## # ... with 132 more rows, and 1 more variable: gdpPercapRel <dbl>
```

I advise that your analyses NEVER rely on

- rows or variables
- being in a specific order.

But it's still true that human beings write the code

- and the interactive development process
- can be much nicer if you reorder the rows of your data
  - as you go along.

Also, once you are preparing tables for human eyeballs,

- it is imperative that you step up and take control of row order.

3.2.4.4 Use `rename()` to rename variables

I am in the awkward life stage of switching from

- camelCase
- to snake_case,

[I (Roger) prefer CamelCase. the underscore in variable names is hard to see.]

So I am vexed by the variable names

- I chose when I cleaned this data years ago.

Let's rename some variables!

```
gtbl %>%
  rename(life_exp = lifeExp, gdp_percap = gdpPercap,
         gdp_percap_rel = gdpPercapRel)
```

```
## # A tibble: 1,704 x 9
##    country  year    pop continent life_exp gdp_percap     gdp canada
##    <fct>   <int>  <dbl> <fct>        <dbl>      <dbl>   <dbl>  <dbl>
##  1 Afghan~  1952 8.43e6 Asia          28.8       779. 6.57e 9 11367.
##  2 Afghan~  1957 9.24e6 Asia          30.3       821. 7.59e 9 12490.
##  3 Afghan~  1962 1.03e7 Asia          32.0       853. 8.76e 9 13462.
##  4 Afghan~  1967 1.15e7 Asia          34.0       836. 9.65e 9 16077.
##  5 Afghan~  1972 1.31e7 Asia          36.1       740. 9.68e 9 18971.
##  6 Afghan~  1977 1.49e7 Asia          38.4       786. 1.17e10 22091.
##  7 Afghan~  1982 1.29e7 Asia          39.9       978. 1.26e10 22899.
##  8 Afghan~  1987 1.39e7 Asia          40.8       852. 1.18e10 26627.
##  9 Afghan~  1992 1.63e7 Asia          41.7       649. 1.06e10 26343.
## 10 Afghan~  1997 2.22e7 Asia          41.8       635. 1.41e10 28955.
## # ... with 1,694 more rows, and 1 more variable: gdp_percap_rel <dbl>
```

I did NOT assign the post-rename object back to `gtbl`

- because that would make the chunks in this tutorial
  - harder to copy/paste and run out of order.

In real life, I would probably assign this back to `gtbl`,

- in a data cleaning and assembly script,
- and proceed with the new variable names.

3.2.4.5 `group_by()` is a mighty weapon

I have found friends and family love to ask seemingly innocuous questions like,

- "which country experienced the sharpest 5-year drop in life expectancy?".

In fact, that is a totally natural question to ask.

But if you are using a language that doesn't know about data,

- it's an incredibly annoying question to answer.

`dplyr` offers powerful tools to solve this class of problem.

- `group_by()` adds extra structure to your dataset
  - – grouping information –
  - which lays the groundwork for computations within the groups.
- `summarize()` takes a dataset with n observations,
  - computes requested summaries,
  - and returns a dataset with 1 observation.
- window functions take a dataset with n observations
  - and return a dataset with n observations.

Combined with the verbs you already know,

- these new tools allow you to solve an extremely diverse
- set of problems with relative ease.

3.2.4.5.1 Counting things up

Let's start with simple counting.

- How many observations do we have per continent?

```
gtbl %>%
  group_by(continent) %>%
  summarize(n_obs = n())
```

```
## # A tibble: 5 x 2
##   continent n_obs
##   <fct>     <int>
## 1 Africa      624
## 2 Americas    300
## 3 Asia        396
## 4 Europe      360
## 5 Oceania      24
```

The `tally()` function is a convenience function for this sort of thing.

```
gtbl %>%
  group_by(continent) %>%
  tally
```

```
## # A tibble: 5 x 2
##   continent     n
##   <fct>     <int>
## 1 Africa      624
## 2 Americas    300
## 3 Asia        396
## 4 Europe      360
## 5 Oceania      24
```

What if we wanted to add the number of unique countries for each continent?

```
gtbl %>%
  group_by(continent) %>%
  summarize(n_obs = n(), n_countries = n_distinct(country))
```

```
## # A tibble: 5 x 3
##   continent n_obs n_countries
##   <fct>     <int>       <int>
## 1 Africa      624          52
## 2 Americas    300          25
```

```
## 3 Asia        396          33
## 4 Europe      360          30
## 5 Oceania      24           2
```

3.2.4.5.2 General summarization

The functions you'll apply within `summarize()`

- include classical statistical summaries,
- like `mean()`, `median()`, `sd()`, and `IQR`.

Remember they are functions that take n inputs

- and distill them down into 1 output.

Although this may be statistically ill-advised,

- let's compute the average life expectancy by continent.

```
gtbl %>%
  group_by(continent) %>%
  summarize(avg_lifeExp = mean(lifeExp))
```

```
## # A tibble: 5 x 2
##   continent avg_lifeExp
##   <fct>           <dbl>
## 1 Africa           48.9
## 2 Americas         64.7
## 3 Asia             60.1
## 4 Europe           71.9
## 5 Oceania          74.3
```

`summarize_each()` applies the same summary function(s)

- to multiple variables.

Let's compute

- average and median life expectancy
- and GDP per capita by continent by year ...
- but only for 1952 and 2007.

```
gtbl %>%
  filter(year %in% c(1952, 2007)) %>%
  group_by(continent, year) %>%
  summarise_each(funs(mean, median), lifeExp, gdpPercap)
```

```
## `summarise_each()` is deprecated.
## Use `summarise_all()`, `summarise_at()` or `summarise_if()` instead.
## To map `funs` over a selection of variables, use `summarise_at()`
```

```
## # A tibble: 10 x 6
## # Groups:   continent [?]
##    continent  year lifeExp_mean gdpPercap_mean lifeExp_median
##    <fct>     <int>        <dbl>          <dbl>          <dbl>
##  1 Africa     1952         39.1          1253.           38.8
##  2 Africa     2007         54.8          3089.           52.9
##  3 Americas   1952         53.3          4079.           54.7
##  4 Americas   2007         73.6         11003.           72.9
##  5 Asia       1952         46.3          5195.           44.9
```

```
##  6 Asia       2007         70.7         12473.           72.4
##  7 Europe     1952         64.4          5661.           65.9
##  8 Europe     2007         77.6         25054.           78.6
##  9 Oceania    1952         69.3         10298.           69.3
## 10 Oceania    2007         80.7         29810.           80.7
## # ... with 1 more variable: gdpPercap_median <dbl>
```

Let's focus just on Asia.

What are the minimum and maximum life expectancies seen by year?

```
gtbl %>%
  filter(continent == "Asia") %>%
  group_by(year) %>%
  summarize(min_lifeExp = min(lifeExp), max_lifeExp = max(lifeExp))
```

```
## # A tibble: 12 x 3
##     year min_lifeExp max_lifeExp
##    <int>       <dbl>       <dbl>
##  1  1952        28.8        65.4
##  2  1957        30.3        67.8
##  3  1962        32.0        69.4
##  4  1967        34.0        71.4
##  5  1972        36.1        73.4
##  6  1977        31.2        75.4
##  7  1982        39.9        77.1
##  8  1987        40.8        78.7
##  9  1992        41.7        79.4
## 10  1997        41.8        80.7
## 11  2002        42.1        82
## 12  2007        43.8        82.6
```

Of course it would be much more interesting

- to see *which* country contributed these extreme observations.
- Is the minimum (maximum) always coming from the same country?
- That's where window functions come in.

3.2.4.5.3 Window functions

Recall that window functions take n inputs

- and give back n outputs.

Here we use window functions based on ranks and offsets.

Let's revisit the worst and best life expectancies in Asia over time,

- but retaining info about *which* country contributes these extreme values.

```
gtbl %>%
  filter(continent == "Asia") %>%
  select(year, country, lifeExp) %>%
  arrange(year) %>%
  group_by(year) %>%
  filter(min_rank(desc(lifeExp)) < 2 | min_rank(lifeExp) < 2)
```

```
## # A tibble: 24 x 3
## # Groups:   year [12]
```

```
##     year country     lifeExp
##    <int> <fct>         <dbl>
##  1  1952 Afghanistan    28.8
##  2  1952 Israel         65.4
##  3  1957 Afghanistan    30.3
##  4  1957 Israel         67.8
##  5  1962 Afghanistan    32.0
##  6  1962 Israel         69.4
##  7  1967 Afghanistan    34.0
##  8  1967 Japan          71.4
##  9  1972 Afghanistan    36.1
## 10  1972 Japan          73.4
## # ... with 14 more rows
```

We see that (min = Agfhanistan, max = Japan)

- is the most frequent result,
- but Cambodia and Israel pop up at least once each
  - as the min or max, respectively.
- That table should make you impatient for our upcoming work
  - on tidying and reshaping data!
- Wouldn't it be nice to have one row per year?

How did that actually work?

- First, I store and view the result
  - including everything but the last `filter()` statement.
- All of these operations are familiar.

```
asia <- gtbl %>%
  filter(continent == "Asia") %>%
  select(year, country, lifeExp) %>%
  arrange(year) %>%
  group_by(year)
asia
```

```
## # A tibble: 396 x 3
## # Groups:   year [12]
##     year country          lifeExp
##    <int> <fct>              <dbl>
##  1  1952 Afghanistan         28.8
##  2  1952 Bahrain             50.9
##  3  1952 Bangladesh          37.5
##  4  1952 Cambodia            39.4
##  5  1952 China               44
##  6  1952 Hong Kong, China    61.0
##  7  1952 India               37.4
##  8  1952 Indonesia           37.5
##  9  1952 Iran                44.9
## 10  1952 Iraq                45.3
## # ... with 386 more rows
```

Now we apply a window function – `min_rank()`.

- Since `asia` is grouped by year,
  - `min_rank()` operates within mini-datasets,
  - each for a specific year.
- Applied to the variable `lifeExp`,

  - `min_rank()` returns the rank of each country's observed life expectancy.
- FYI, the `min` part just specifies how ties are broken.
  - Here is an explicit peek at these within-year life expectancy ranks,
  - in both the (default) ascending and descending order.

```
asia %>%
  mutate(le_rank = min_rank(lifeExp),
         le_desc_rank = min_rank(desc(lifeExp)))
```

```
## # A tibble: 396 x 5
## # Groups:   year [12]
##     year country          lifeExp le_rank le_desc_rank
##    <int> <fct>              <dbl>   <int>        <int>
##  1  1952 Afghanistan         28.8       1           33
##  2  1952 Bahrain             50.9      25            9
##  3  1952 Bangladesh          37.5       7           27
##  4  1952 Cambodia            39.4       9           25
##  5  1952 China               44        16           18
##  6  1952 Hong Kong, China    61.0      31            3
##  7  1952 India               37.4       5           29
##  8  1952 Indonesia           37.5       6           28
##  9  1952 Iran                44.9      17           17
## 10  1952 Iraq                45.3      18           16
## # ... with 386 more rows
```

You can understand the original `filter()` statement now:

```
filter(min_rank(desc(lifeExp)) < 2 | min_rank(lifeExp) < 2)
```

These two sets of ranks are formed,

- within year group,
  - and `filter()` retains rows with rank less than 2,
  - which means ... the row with rank = 1. Since we do for ascending and descending ranks,
  - we get both the min and the max.

If we had wanted just the min OR the max,

- an alternative approach using `top_n()`
- would have worked.

```
gtbl %>%
  filter(continent == "Asia") %>%
  select(year, country, lifeExp) %>%
  arrange(year) %>%
  group_by(year) %>%
  #top_n(1)               ## gets the min
  top_n(1, desc(lifeExp)) ## gets the max
```

```
## # A tibble: 12 x 3
## # Groups:   year [12]
##     year country     lifeExp
##    <int> <fct>         <dbl>
##  1  1952 Afghanistan    28.8
##  2  1957 Afghanistan    30.3
##  3  1962 Afghanistan    32.0
##  4  1967 Afghanistan    34.0
##  5  1972 Afghanistan    36.1
```

```
##  6  1977 Cambodia       31.2
##  7  1982 Afghanistan    39.9
##  8  1987 Afghanistan    40.8
##  9  1992 Afghanistan    41.7
## 10  1997 Afghanistan    41.8
## 11  2002 Afghanistan    42.1
## 12  2007 Afghanistan    43.8
```

3.2.4.5.4 Grand Finale

So let's answer that "simple" question:

- which country experienced the sharpest 5-year drop in life expectancy?

Recall that this excerpt of the Gapminder data

- only has data every five years,
  - e.g. for 1952, 1957, etc.
- So this really means looking at life expectancy changes
  - between adjacent timepoints.

At this point, that's just too easy,

- so let's do it by continent while we're at it.

```
gtbl %>%
  group_by(continent, country) %>%
  select(country, year, continent, lifeExp) %>%
  mutate(le_delta = lifeExp - lag(lifeExp)) %>%
  summarize(worst_le_delta = min(le_delta, na.rm = TRUE)) %>%
  filter(min_rank(worst_le_delta) < 2) %>%
  arrange(worst_le_delta)
```

```
## # A tibble: 5 x 3
## # Groups:   continent [5]
##   continent country     worst_le_delta
##   <fct>     <fct>                <dbl>
## 1 Africa    Rwanda             -20.4
## 2 Asia      Cambodia            -9.10
## 3 Americas  El Salvador         -1.51
## 4 Europe    Montenegro          -1.46
## 5 Oceania   Australia            0.170
```

Ponder that for a while.

- The subject matter and the code.
- Mostly you're seeing what genocide looks like
  - in dry statistics on average life expectancy.

Break the code into pieces, starting at the top,

- and inspect the intermediate results.

That's certainly how I was able to *write* such a thing.

These commands do not leap fully formed out of anyone's forehead

- they are built up gradually,
  - with lots of errors and refinements along the way.
- I'm not even sure it's a great idea

    - to do so much manipulation in one fell swoop.

Is the statement above really hard for you to read?

- If yes, then by all means break it into pieces
    - and make some intermediate objects.
- Your code should be easy to write and read when you're done.

In later tutorials,

- we'll explore more of `dplyr`, such as operations based on two datasets.

3.2.4.6 Links

Jenny Bryan Stat 545

`dplyr` official stuff

- package home on CRAN
    - note there are several vignettes, with the introduction being the most relevant right now
    - the one on window functions will also be interesting to you now
- development home on GitHub
- tutorial HW delivered (note this links to a DropBox folder) at useR! 2014 conference

Blog post Hands-on dplyr tutorial for faster data manipulation in R by Data School, that includes a link to an R Markdown document and links to videos

Cheatsheet I made for `dplyr` join functions (not relevant yet but soon)