Basic Constructs in R

R structures, and the apply class of functions

Spencer Lourens

November 05 2016

Data Wrangling! (Using dplyr, tidyr)

- **Data wrangling**: The practical task of transforming the *layout*, *substance*, and *display* of your data - Garrett Grolemund, RStudio

Data Wrangling! (Using dplyr, tidyr)

- **Data wrangling**: The practical task of transforming the *layout*, *substance*, and *display* of your data - Garrett Grolemund, RStudio
- **Data wrangling** can be broken into three basic domains: data manipulation, data tidying (for analysis/visualization), and data visualization

Data Wrangling! (Using dplyr, tidyr)

- **Data wrangling**: The practical task of transforming the *layout*, *substance*, and *display* of your data - Garrett Grolemund, RStudio
- **Data wrangling** can be broken into three basic domains: data manipulation, data tidying (for analysis/visualization), and data visualization
- *Data manipulation* - Change the variables, their values, or the sample units used for analysis (aggregation?)

Data Wrangling! (Using dplyr, tidyr)

- **Data wrangling**: The practical task of transforming the *layout*, *substance*, and *display* of your data - Garrett Grolemund, RStudio
- **Data wrangling** can be broken into three basic domains: data manipulation, data tidying (for analysis/visualization), and data visualization
- *Data manipulation* - Change the variables, their values, or the sample units used for analysis (aggregation?)
- *Data tidying* - Change the LAYOUT of the data - wide, long, etc., most of the time for software use

Data Wrangling! (Using dplyr, tidyr)

- **Data wrangling**: The practical task of transforming the *layout*, *substance*, and *display* of your data - Garrett Grolemund, RStudio
- **Data wrangling** can be broken into three basic domains: data manipulation, data tidying (for analysis/visualization), and data visualization
- *Data manipulation* - Change the variables, their values, or the sample units used for analysis (aggregation?)
- *Data tidying* - Change the LAYOUT of the data - wide, long, etc., most of the time for software use
- *Data visualization* - Make use of our visual way of thinking by representing data in a visual format

Preliminaries

- The dplyr package creates another structure, called a tbl (table, or as Hadley says, tibble)

Preliminaries

- The dplyr package creates another structure, called a tbl (table, or as Hadley says, tibble)
- A tbl is a dataframe

Preliminaries

- The dplyr package creates another structure, called a tbl (table, or as Hadley says, tibble)
- A tbl is a dataframe
  - that works seemlessly with dplyr functions

Preliminaries

- The dplyr package creates another structure, called a tbl (table, or as Hadley says, tibble)
- A tbl is a dataframe
  - that works seemlessly with dplyr functions
  - that optimizes display when printed without printing too much and taking up unnecessary cache memory (slows down your R session)

Preliminaries

- The dplyr package creates another structure, called a tbl (table, or as Hadley says, tibble)
- A tbl is a dataframe
  - that works seemlessly with dplyr functions
  - that optimizes display when printed without printing too much and taking up unnecessary cache memory (slows down your R session)
- The pipe (%>%) operator

Preliminaries

- The dplyr package creates another structure, called a tbl (table, or as Hadley says, tibble)
- A tbl is a dataframe
  - that works seemlessly with dplyr functions
  - that optimizes display when printed without printing too much and taking up unnecessary cache memory (slows down your R session)
- The pipe (%>%) operator
  - allows more concise, perhaps narrative-looking production of code

Piping example

- Consider the two approaches below

```r
x <- c(1,3,4,1,4,10,19,2.5,1)
x %>% mean(na.rm = T)
```

```
## [1] 5.055556
```

```r
mean(x)
```

```
## [1] 5.055556
```

- Same result!

Piping example (Cont'd)

- Piping really shines when you have many nested calls as in toy example below

Piping example (Cont'd)

- Piping really shines when you have many nested calls as in toy example below
- The following are equivalent:
  - reportResult(groupResult(calculateResult(df))
  - df %>% calculateResult() %>% groupResult() %>% reportResult()

Piping example (Cont'd)

- Piping really shines when you have many nested calls as in toy example below
- The following are equivalent:
  - reportResult(groupResult(calculateResult(df))
  - df %>% calculateResult() %>% groupResult() %>% reportResult()
- Notice that the second line with piping is more natural to read from left to right, similar to how we read English
- Very helpful in more complicated settings

Example datasets

- It may be helpful to go back over these notes in your own R session - we'll test a lot of dplyr functionality on different datasets

Example datasets

- It may be helpful to go back over these notes in your own R session - we'll test a lot of dplyr functionality on different datasets

Example datasets

- It may be helpful to go back over these notes in your own R session - we'll test a lot of dplyr functionality on different datasets
- Functions from the dplyr package to demonstrate: select, filter, mutate,

Example datasets

- It may be helpful to go back over these notes in your own R session - we'll test a lot of dplyr functionality on different datasets
- Functions from the dplyr package to demonstrate: select, filter, mutate,
- We will illustrate use of these functions on the flights data set from package **nycflights13**, previewed on the following slide, and the storms data set from package **EDAWR**, on the subsequent slide.

Dataset flights

- The actual dimension of flights is 336776 by 19

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
2013	1	1	517	515	2	830
2013	1	1	533	529	4	850
2013	1	1	542	540	2	923
2013	1	1	544	545	-1	1004
2013	1	1	554	600	-6	812
2013	1	1	554	558	-4	740
2013	1	1	555	600	-5	913
2013	1	1	557	600	-3	709
2013	1	1	557	600	-3	838
2013	1	1	558	600	-2	753

Dataset storms

- The actual dimension of flights is 6 by 4

storm	wind	pressure	date
Alberto	110	1007	2000-08-03
Alex	45	1009	1998-07-27
Allison	65	1005	1995-06-03
Ana	40	1013	1997-06-30
Arlene	50	1010	1999-06-11
Arthur	45	1010	1996-06-17

The dplyr::select function

- select: Select/rename variables by name
- **select(.data, ..., .dots)** - .data: a tbl, ...: comma separated list of unquoted expressions, .dots: Use select_() to do standard evaluation (using strings)

```
flights %>% select(year, month)
```

year	month
2013	1
2013	1
2013	1
2013	1

The dplyr::select function

- We often have a fixed vector of string variables we would like to keep in the analysis
- Use select_() with .dots argument!

```
flights %>% select_(.dots = c("year", "month", "dep_time",
                              "dep_delay", "arr_time",
                              "arr_delay", "flight"))
```

year	month	dep_time	dep_delay	arr_time	arr_delay	flight
2013	1	517	2	830	11	1545
2013	1	533	4	850	20	1714
2013	1	542	2	923	33	1141
2013	1	544	-1	1004	-18	725
2013	1	554	-6	812	-25	461

The dplyr::select function

- Sometimes we might only want to remove a variable or variables

```
flights %>% select(-year)
flights %>% select_(.dots = c("-year", "-origin", "-day"))
flights %>% select(-sched_dep_time:-arr_time)
```

Special functions for dplyr::select

- Several variables in your dataset may have a similar naming convention, i.e. variablet1, variablet2, etc.
- Perhaps we only want to extract all of this information in one swoop, how do we do it?
- Use ends_with, starts_with, contains, matches, num_range, one_of....

```
## Select variables ending with time and the year/month
flights %>% select(year, month, ends_with("time"))
## Select variables starting with arr and the year/month
flights %>% select(year, month, starts_with("arr"))
```

year	month	dep_time	sched_dep_time	arr_time
2013	1	517	515	830
2013	1	533	529	850
2013	1	542	540	923
2013	1	544	545	1004
2013	1	554	600	812

The dplyr::filter function

- Sometimes we need to include only observations which satisfy some criterion for analysis
  - Include only cases
  - Include only those above certain exposure level
  - Include only those with positive survival times
- The filter function comes to the rescue!

The dplyr::filter function

- filter: Return rows with matching conditions
- **filter(.data, ...)** - .data: a tbl, ...: logical predicates. Multiple conditions are conmbined with &

```
storms %>% filter(pressure == 1010)
```

storm	wind	pressure	date
Arlene	50	1010	1999-06-11
Arthur	45	1010	1996-06-17

```
storms %>% filter(storm == 'Alberto' | pressure > 1009)
```

storm	wind	pressure	date
Alberto	110	1007	2000-08-03
Ana	40	1013	1997-06-30
Arlene	50	1010	1999-06-11
Arthur	45	1010	1996-06-17

The dplyr::filter function

- Can use all comparisons, logical expressions in filter(), filter_(). The filter_() function takes string arguments rather than explicit expressions
- Logical expressions: &, |, !, any, all, . . .
- Comparisons: <, >, ==, <=, >=, !=, %in%, is.na, !is.na, . . .

```
flights %>% filter(!is.na(air_time))
```

year	month	day	dep_time	sched_dep_time
2013	1	1	517	515
2013	1	1	533	529
2013	1	1	542	540
2013	1	1	544	545

The dplyr::summarise function

- summarise: Summarise multiple values to a single value.
- **summarise(.data, ...)** - .data: a tbl, ...: name-value pairs of summary functions. For example, mean(), sd(), etc.

```
storms %>% summarise(mean_wind = mean(wind),
                     mean_pressure = mean(pressure))
```

Source: local data frame [1 x 2]

mean_wind mean_pressure (dbl) (dbl) 1 59.16667 1009