Grammar of data

dplyr

Bjarki Þór Elvarsson and Einar Hjörleifsson

Marine Research Institute

Working with data

A Reformat a variable (e.g. as factors or dates)

B Split one variable into two or more

C Join two or more variables together

D Create new variables based on calculated results

E Create variables from bits of text

F Rename variables

G Create summaries

dplyr and tidyr

- dplyr and tidyr are a set of tools for a common set of problems connected to aggregates or summaries of data.
- Similar to ggplot2 they feature a Domain Specific Language (DSL) specially designed for data summaries.
- Developed by Hadley Wickam, the creator ggplot2 and other useful tools.

Summarising data for groups

Commonly, when collating summaries by group, one wants to:

- **Split** up a big data structure into homogeneous pieces,
- **Apply** a function to each piece
- **Combine** all the results back together.

For example, one might want to

- fit the same model each patient subsets of a data frame
- quickly calculate summary statistics for each group
- perform group-wise transformations like scaling or standardising

One table verbs

- filter: keep rows matching criteria
- select: pick columns by name
- arrange: order the rows according to a variable
- mutate: add new variables
- summarise: reduce variables to values

Structure

- First argument is a data frame
- Always return a data frame
- Subsequent arguments say what to do with data frame
- (Never modify in place)

filter

- select rows that satisfy a certain condition
- input dataframe/database table and bolean condition

df

color	value
blue	1
black	2
blue	3
blue	4
black	5

color	value
blue	1
blue	3
blue	4

```
filter(df, color == "blue")
```

select

- select only certain columns
- input dataframe/database table and column names
- allows negative index of column names and allows renaming

df

color	value
blue	1
black	2
blue	3
blue	4
black	5

color
blue
black
blue
blue
black

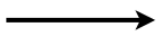
```
select(df, color)
```

arrange

- arranges by certain columns
- input dataframe/database table and column names to arrange by
- defaults to ascending order but can arrange in descending by writing desc around the column name

df

color	value
4	1
1	2
5	3
3	4
2	5



color	value
1	2
2	5
3	4
4	1
5	3

```
arrange(df, color)
```

mutate

- Adds and modifies columns
- input dataframe and column names with modifying formulas
- column are created using R commands

df

color	value
blue	1
black	2
blue	3
blue	4
black	5

color	value	double
blue	1	2
black	2	4
blue	3	6
blue	4	8
black	5	10

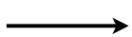
```
mutate(df, double = 2 * value)
```

summarise

- Creates summaries from tabular data
- input dataframe and column names representing the summaries
- column are created using R commands

df

color	value
blue	1
black	2
blue	3
blue	4
black	5



total
15

```
summarise(df, total = sum(value))
```

Group verbs

- group_by: Group data into rows with the same value of (a) particular variable(s)

```
minke <- group_by(minke,sex)
```

- ungroup: Remove grouping information from data frame

```
minke <- ungroup(minke)
```

Grouped summaries

df

color	value
blue	1
black	2
blue	3
blue	4
black	5

color	total
blue	8
black	7

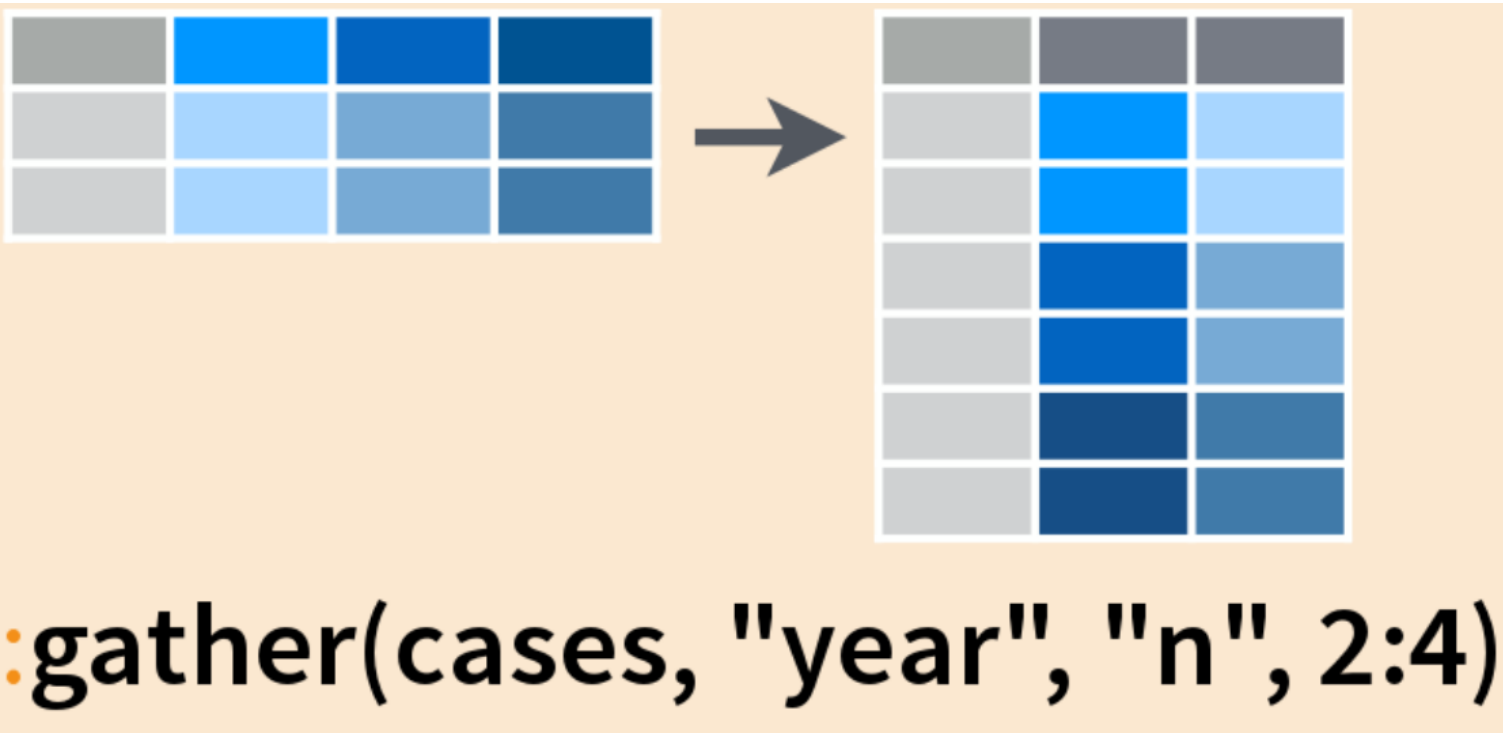
```
by_color <- group_by(df, color)
summarise(by_color, total = sum(value))
```

Reshape verbs

- gather: Gather columns into rows
- spread: Spread rows into columns
- separate: Separate one column into many
- unite: Unite several columns into one

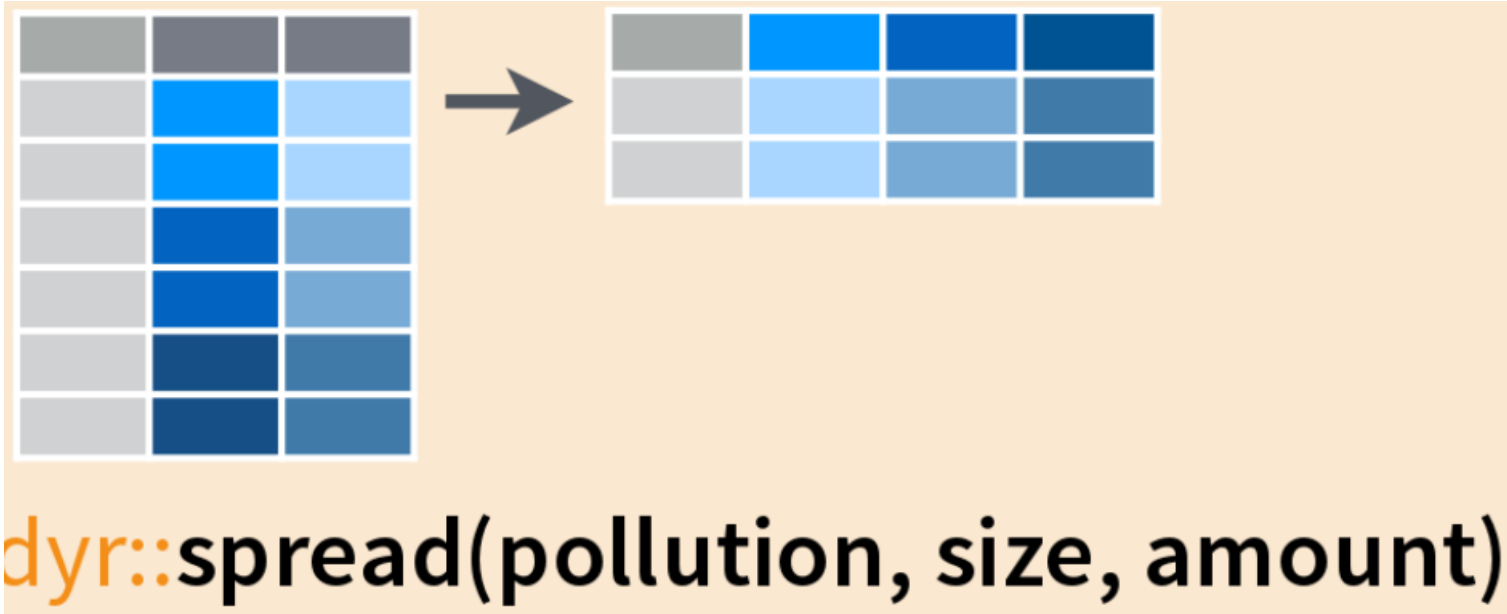
Gather

- Takes data from a wide format (i.e. human readable) to a long format (computer readable.
- Inputs are data, key columns and value columns
- Gather allows the negative indexing for both key and value column names



Spread

- Takes data from a long format to a wide format).
- Inputs are data, key column (i.e. new column names) and value column



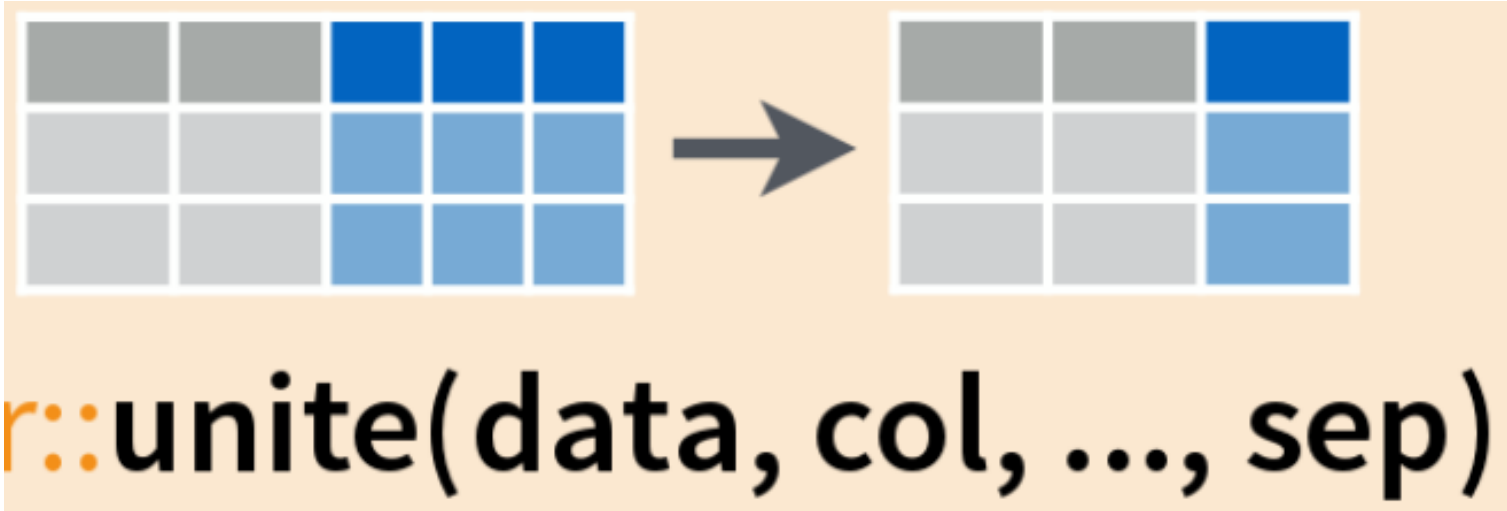
Separate

- Splits a column into two (or more) columns
- Inputs are data, column to be split, name of the new columns (as characters) and the splitting character

```
separate(storms, date, c("y", "m", "d"))
```

Unite

- Unites two or more columns
- Inputs are data, name of the new (united) column, names of columns to be united and separating character



Joining data together

One can join together two data.frames in a number of ways

```
inner_join(x,y)    ## only rows where there is a
                   ## match between x and y
left_join(x,y)     ## keep x add matching data from y
right_join(x,y)    ## keep y add matching data from x
semi_join(x,y)     ## keep only data from x
                   ## that matches y
anti_join(x,y)     ## keep only data from x
                   ## that doesn't match y
full_join(x,y)     ## keep all x and y
                   ## (all NA's where no match)
```

Chaining expressions together

- In R one can apply functions to data:

```
avg.l <- mean(minke$length)
l2 <- avg.l^2
```

- One also chain this together:

```
l2 <- mean(minke$length)^2
```

Chaining expressions together

All this can quickly become cumbersome and hard to read (and modify):

```
summarise(group_by(filter(minke,!is.na(weight)),
                   sex),num.whale=n(),
          m.weight = mean(weight))
```

What does this command do?

The %>% operator

Operations can however chained using the %>% operator from dplyr

```
minke %>%
  filter(!is.na(weight)) %>%
  group_by(sex) %>%
  summarise(num.whale = n(),
            m.weight = mean(weight))

## hint think of %>% as 'then'
```

The %>% operator pushes the output from the first command as the first input to the next command

Further reading

- https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf
- https://cran.rstudio.com/web/packages/dplyr/-vignettes/introduction.html
- http://vita.had.co.nz/papers/tidy-data.pdf
- http://www.jvcasillas.com/tidyr_tutorial/
- http://stackoverflow.com/questions/3505701/r-grouping-functions-sapply-vs-lapply-vs-apply-vs-tapply-vs-by-vs-aggrega

Excercise 3

Create a new script, 'Ex3.R' and write code that produces:

- The number of whales caught each year
- The proportion caught of which are females each year
- Calculate the mean length and age along with standard deviation grouped by maturity
- Using $\%>\%$ and spread, calculate number of whales caught by area (rows) and year (columns)

Useful string operations

The 'stringr' package adds a number of string operations:

```
str_c()          ## glues strings together
str_length()     ## measure the length of a string
str_sub()        ## select parts of the string
str_str() <-     ## assign parts of the string with new values
str_dup()        ## duplicates string
str_trim()       ## removes trailing white space
str_pad()        ## adds whitespace
sprintf()        ## creates new strings using wildcard replaceme
```

Find and replace

Stringr also does find and replace:

```
str_detect()       ## finds matching string location
str_subset()       ## finds matching string values
str_locate()       ## finds the first match and returns
                   ## location within the string
str_locate_all()   ## same as above but all matches
str_extract()      ## extract first string that matches
str_extract_all()  ## extract all strings that match
str_replace()      ## replace matching string
str_split()        ## splits text according to a split char
```

Regular expressions

Simple useful search expression:

```
[a-z]          ## match all lower case letters
[A-Z]          ## match all upper case letters
[0-9]          ## match all numbers
^              ## don't match
+              ## preceding match will be matched once or more
.              ## everything except the empty string
*              ## everything
$              ## end of the line
|              ## or
()             ## grouping
```

Working with dates

Class excercise