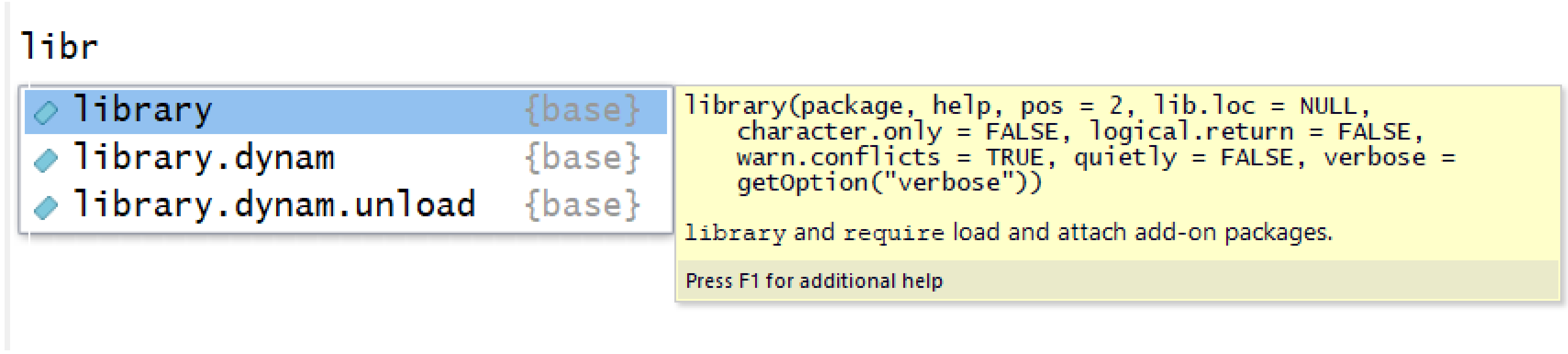
Workshop 3:
The tidyverse and beyond

- Send an SOS to the world

Brendan Palmer,
Statistics & Data Analysis Unit,
Clinical Research Facility - Cork

Levels of help within RStudio

1. Let R help you write your code using Tab



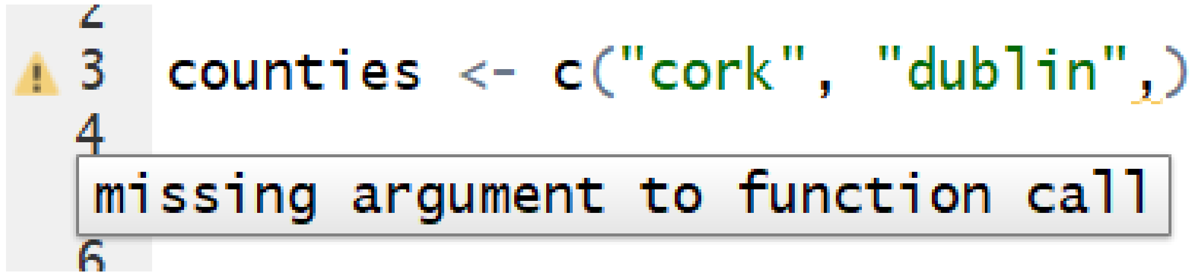
```
libr
library                 {base}
library.dynam           {base}
library.dynam.unload    {base}
```

```
library(package, help, pos = 2, lib.loc = NULL,
    character.only = FALSE, logical.return = FALSE,
    warn.conflicts = TRUE, quietly = FALSE, verbose =
    getOption("verbose"))
```

library and require load and attach add-on packages.

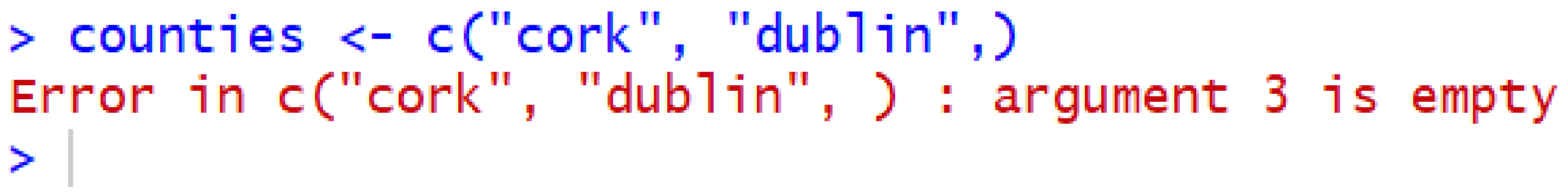
Press F1 for additional help

2. Hover over the error symbols to identify what the error is



```
counties <- c("cork", "dublin",)
```

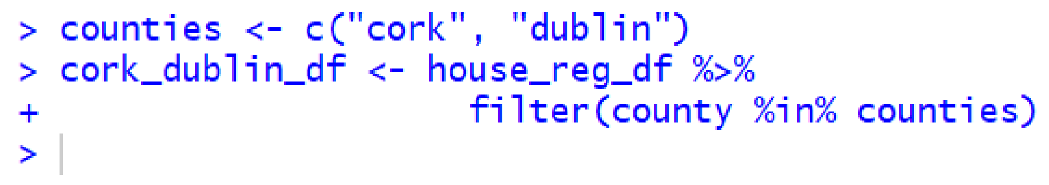
missing argument to function call



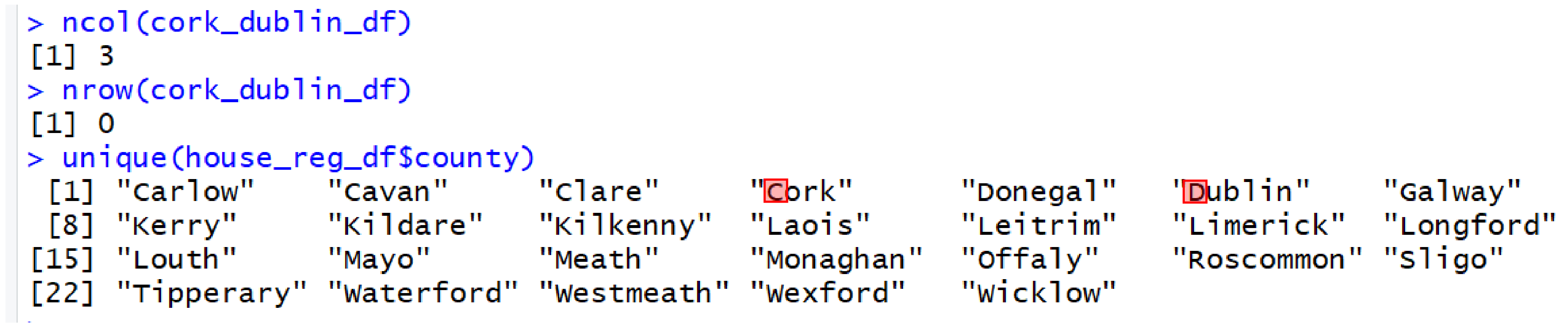
```
> counties <- c("cork", "dublin",)
Error in c("cork", "dublin", ) : argument 3 is empty
>
```

Levels of help within RStudio

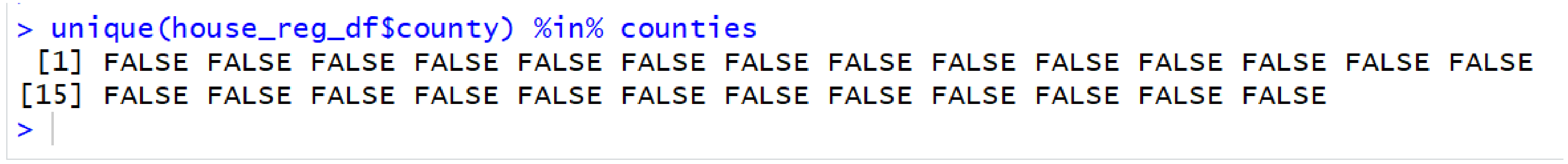
3. Watch out for capitalisation and naming errors as the code might run, but all the arguments supplied may return "FALSE"



```
> counties <- c("cork", "dublin")
> cork_dublin_df <- house_reg_df %>%
+                   filter(county %in% counties)
> 
```



```
> ncol(cork_dublin_df)
[1] 3
> nrow(cork_dublin_df)
[1] 0
> unique(house_reg_df$county)
 [1] "Carlow"    "Cavan"     "Clare"     "Cork"      "Donegal"   "Dublin"    "Galway"   
 [8] "Kerry"     "Kildare"   "Kilkenny"  "Laois"     "Leitrim"   "Limerick"  "Longford" 
[15] "Louth"     "Mayo"      "Meath"     "Monaghan"  "Offaly"    "Roscommon" "Sligo"    
[22] "Tipperary" "Waterford" "Westmeath" "Wexford"   "Wicklow"  
```

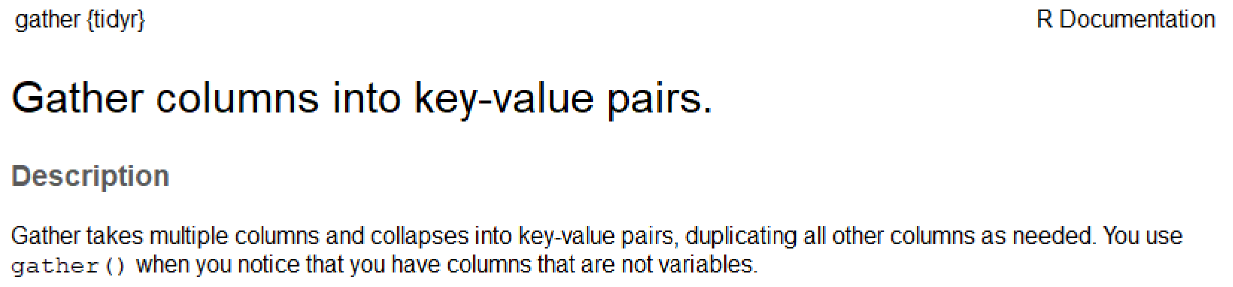


```
> unique(house_reg_df$county) %in% counties
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[15] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> 
```

Levels of help within RStudio

4. `> ?gather`

   - If you're unsure about a function, place the question mark before it to retrieve some help documentation



gather {tidyr} R Documentation

Gather columns into key-value pairs.

Description

Gather takes multiple columns and collapses into key-value pairs, duplicating all other columns as needed. You use `gather()` when you notice that you have columns that are not variables.

   - describes the various arguments to the function
   - provides useful examples to guide you through common operations

```
# get first observation for each Species in iris data -- base R
mini_iris <- iris[c(1, 51, 101), ]
# gather Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
gather(mini_iris, key = flower_att, value = measurement,
       Sepal.Length, Sepal.Width, Petal.Length, Petal.Width)
```

Worksheet

Open script1_ws3_correct_the_errors.R

Package Cheatsheets

- Cheatsheets are available for commonly used packages
  - c.f. last week we explored the baseR cheatsheet

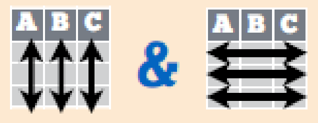
- Useful for quick reference to the most commonly used functions for that package
  - https://www.rstudio.com/resources/cheatsheets/

- Open the data transformation cheatsheet

Data Transformation with dplyr Cheat Sheet

RStudio

dplyr functions work with pipes and expect **tidy data**. In tidy data:



Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

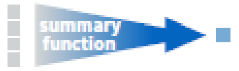
pipes

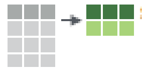
x %>% f(y)

becomes f(x, y)

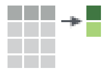
Summarise Cases

These apply **summary functions** to columns to create a new table. Summary functions take vectors as input and return one value (see back).





summarise(.data, ...)
Compute table of summaries. Also **summarise_()**.
summarise(mtcars, avg = mean(mpg))



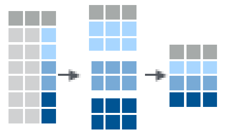
count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in ... Also **tally()**.
count(iris, Species)

Variations

- **summarise_all()** - Apply funs to every column.
- **summarise_at()** - Apply funs to specific columns.
- **summarise_if()** - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to created a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.



```
mtcars %>%
 group_by(cyl) %>%
  summarise(avg = mean(mpg))
```

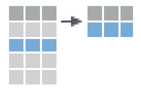
group_by(.data, ..., add = FALSE)
Returns copy of table grouped by ...
g_iris <- group_by(iris, Species)

ungroup(x, ...)
Returns ungrouped copy of table.
ungroup(g_iris)

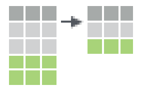
Manipulate Cases

Extract Cases

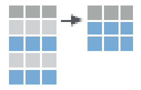
Row functions return a subset of rows as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.



filter(.data, ...)
Extract rows that meet logical criteria. Also **filter_()**. *filter(iris, Sepal.Length > 7)*

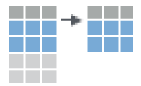


distinct(.data, ..., .keep_all = FALSE)
Remove rows with duplicate values. Also **distinct_()**. *distinct(iris, Species)*



sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame())
Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame())
Randomly select size rows.
sample_n(iris, 10, replace = TRUE)



slice(.data, ...)
Select rows by position. Also **slice_()**.
slice(iris, 10:15)

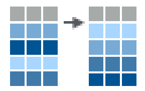
top_n(*x, n, wt*)
Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

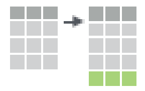
See **?base::logic** and **?Comparison** for help.

Arrange Cases



arrange(.data, ...)
Order rows by values of a column (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

Add Cases

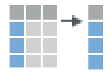


add_row(.data, ..., .before = NULL, .after = NULL)
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

Extract Variables

Column functions return a set of columns as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.



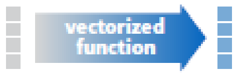
select(.data, ...)
Extract columns by name. Also **select_if()**
select(iris, Sepal.Length, Species)

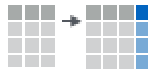
Use these helpers with select(),
e.g. *select(iris, starts_with("Sepal"))*

contains(match)	**num_range**(prefix, range)	:, e.g. mpg:cyl
ends_with(match)	**one_of**(...)	-, e.g. -Species
matches(match)	**starts_with**(match)	**everything()**

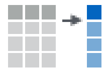
Make New Variables

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

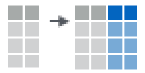




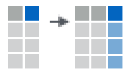
mutate(.data, ...)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)



transmute(.data, ...)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

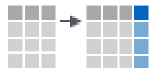


mutate_all(.tbl, .funs, ...)
Apply funs to every column. Use with **funs()**. *mutate_all(faithful, funs(log(.), log2(.)))*

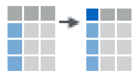


mutate_at(.tbl, .cols, .funs, ...)
Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

mutate_if(.tbl, .predicate, .funs, ...)
Apply funs to all columns of one type. Use with **funs()**.
mutate_if(iris, is.numeric, funs(log(.)))



add_column(.data, ..., .before = NULL, .after = NULL)
Add new column(s).
add_column(mtcars, new = 1:32)

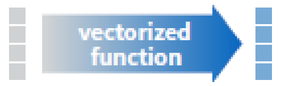


rename(.data, ...)
Rename columns.
rename(iris, Length = Sepal.Length)

Vectorized Functions

to use with mutate()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.



Offsets

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

Cumulative Aggregates

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

Rankings

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

Math

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons

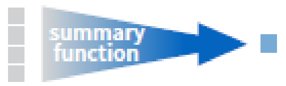
Misc

dplyr::**between()** - x >= left & x <= right
dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

to use with summarise()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.



Counts

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

Location

mean() - mean, also **mean(!is.na())**
median() - median

Logicals

mean() - Proportion of TRUE's
sum() - # of TRUE's

Position/Order

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

Rank

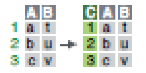
quantile() - nth quantile
min() - minimum value
max() - maximum value

Spread

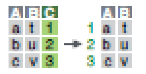
IQR() - Inter-Quartile Range
mad() - mean absolute deviation
sd() - standard deviation
var() - variance

Row names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.



rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

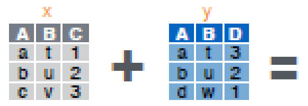


column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

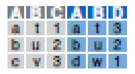
Also **has_rownames(), remove_rownames()**

Combine Tables

Combine Variables

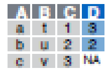


Use **bind_cols()** to paste tables beside each other as they are.

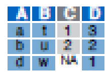


bind_cols(…**)**
Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

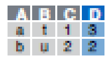
Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.



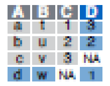
left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join matching values from y to x.



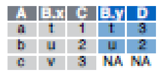
right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join matching values from x to y.



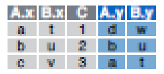
inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join data. Retain only rows with matches.



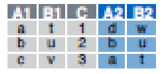
full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join data. Retain all values, all rows.



Use **by = c("col1", "col2")** to specify the column(s) to match on.
left_join(x, y, by = "A")

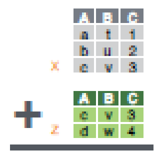


Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

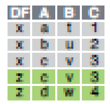


Use **suffix** to specify suffix to give to duplicate column names.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

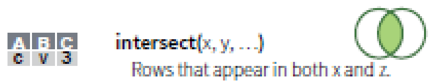
Combine Cases



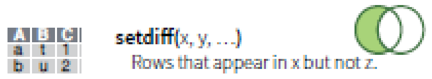
Use **bind_rows()** to paste tables below each other as they are.



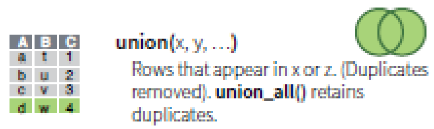
bind_rows(…, .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)



intersect(x, y, …)
Rows that appear in both x and z.



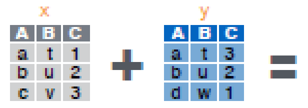
setdiff(x, y, …)
Rows that appear in x but not z.



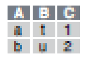
union(x, y, …)
Rows that appear in x or z. (Duplicates removed). **union_all()** retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

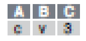
Extract Rows



Use a "**Filtering Join**" to filter one table against the rows of another.



semi_join(x, y, by = NULL, …)
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.



anti_join(x, y, by = NULL, …)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

Package Vignettes

- A vignette is a long-form guide to your package

- Before R 3.0.0, vignettes were standard pdfs
  - The development of RMarkdown has made vignette building and navigation more accessible

- A vignette should divide functions into useful categories, and demonstrate how to coordinate multiple functions to solve problems (but this may not always the case)

- You can see all the installed vignettes with; `browseVignettes()`

- Try it!
  - Click on some hyperlinks to explore the content

Package Webpages

- Many packages are one offs;
  - developed by individuals/labs to solve specific problems
    - once funding expires, package development ends
      - may get released but never updated

- Commonly used packages are updated regularly;
  - new versions released periodically
    - dedicated webpages
      - e.g. lets now explore http://dplyr.tidyverse.org

Worksheet

ws3_script2_help_is_at_hand_.R

Stackoverflow

- Online community to learn, share and improve programming knowledge

- https://stackoverflow.com/

Cross Validated

- Statistics, data analyses, data mining and visualisation

- https://stats.stackexchange.com/

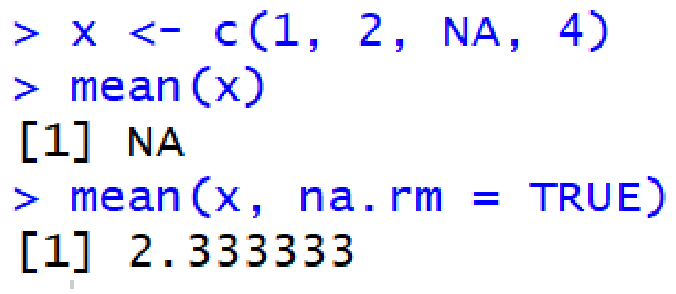
Missing values

- if you apply a calculation to a vector with missing values, the output will be a missing value

1 + 2 + 3 = 6

1 + 2 + NA = NA

- in simple scenarios, NA's can be removed in advance of the calculation with na.rm argument



```
> x <- c(1, 2, NA, 4)
> mean(x)
[1] NA
> mean(x, na.rm = TRUE)
[1] 2.333333
```

Implicit versus explicit missing data

- Implicit
    - The absence of a presence
    - simply not present in the data

```
x <- c(1, 2, 4)
```

- Explicit
    - The presence of an absence
    - flagged with NA

```
x <- c(1, 2, NA, 4)
```

- For each data set you will need to determine to nature of the missing data to decide how to proceed
    - remove
    - impute

Worksheet

Open ws3_script3_missing_data.R

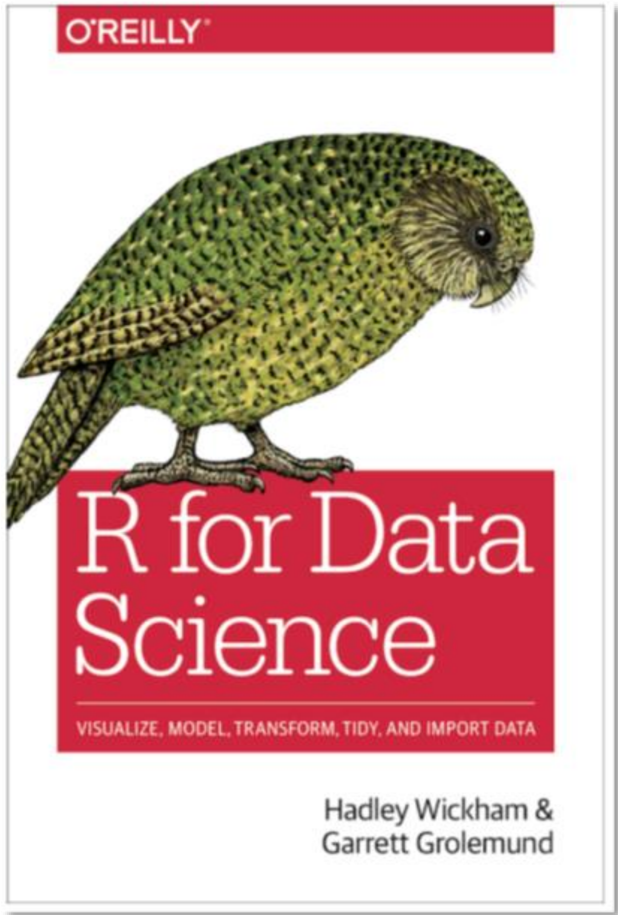
Introductory R Workshops

~~**Week 1 (13th February):**~~
~~**Take a parachute and jump (into the tidyverse)**~~
~~- tidying and visualisation of NGS data using sample R scripts~~

~~**Week 2 (20th February):**~~
~~**We built this software on base R code**~~
~~- overview and structure of R syntax~~

~~**Week 3 (27th February):**~~
~~**Sending an SOS to the world**~~
~~- how to identify with errors in your code and get help~~

Week 4 (6th March):
It's the end of base R as you know it
- introduction to the tidyverse packages tidyr and dplyr


O'REILLY
R for Data Science
VISUALIZE, MODEL, TRANSFORM, TIDY, AND IMPORT DATA
Hadley Wickham & Garrett Grolemund