

Grouping, summarising and joining data

Melbourne Statistical Consulting Platform

University of Melbourne

April 2024

Image by Allison Horst
([Twitter](#)).

dplyr : go wrangling



dplyr verbs

You've seen:

- `summarise()`
- `select()` / `rename()`
- `mutate()` / `transmute()`
- `filter()` / `slice()`
- `arrange()`

Up next:

- more about `group_by()` / `ungroup()`
- apply the same operation to multiple columns using `across()`
- operating rowwise across columns (e.g. mean of Item1, Item2, Item3 for each row) using `rowwise()` and `c_across()`

Refresher: reducing many rows to one summary row

```
airport_screening %>%  
  summarise(  
    n = n(),  
    n_missing = sum(is.na(period_stay)),  
    mean_stay = mean(period_stay, na.rm = TRUE))
```

`summarise()` looks a lot like `mutate()`, but the resulting data frame only has one row, and only the columns you've created.

```
# A tibble: 1 × 3  
      n n_missing mean_stay  
  <int>   <int>   <dbl>  
1    200       0     159
```

Refresher: useful R functions for summarising data

- `sum()`
- `mean()`, `sd()`
- `median()`, `quantile()`, `min()`, `max()`
- `rank()`
- `n()`
- `sum(!is.na())` (counts number of non-missing observations)

Most built-in R functions deal conservatively with missing data: if there is even a single missing value, the function will return `NA`.

Most of the functions on the left accept the option `na.rm = TRUE` to exclude missing observations.

Grouped summaries

Combine `summarise()` with `group_by()` to produce one summary row per group, which is usually more interesting. You can group by multiple variables:

```
airport_screening %>%  
  group_by(arrival_port, passenger_crew) %>%  
  summarise(  
    n = n(),  
    mean_trips = mean(number_trips, na.rm = TRUE),  
    mean_stay = mean(period_stay, na.rm = TRUE),  
    prop_BRM = mean(BRM, na.rm = TRUE)  
  ) %>%  
  ungroup()
```

After calling `group_by()`, don't forget to `ungroup()`. The resulting data frame is still grouped (note warning message)!

``summarise()`` has grouped output by 'arrival_port'. You can override using the ``.groups`` argument.

```
# A tibble: 10 × 6  
  arrival_port passenger_crew      n mean_trips mean_stay prop_BRM  
    <chr>         <chr>    <int>    <dbl>    <dbl>    <dbl>  
1 ADL           P         8      23.8     392.     0.625  
2 BNE           C         3     314.      3.67      0  
3 BNE           P        24      21.9     181.     0.333  
4 DRW           P         5       8.4      27       0.4  
5 MEL           P        61      24.3     156.     0.115  
6 OOL           P         5       8.6     619       0.2  
7 PER           C         1      36       52       0  
8 PER           P         7      19.6     132.     0.143  
9 SYD           C         4     354.      7.25      0  
10 SYD          P        82      24.3     129.     0.146
```

Other uses of grouped data frames

`group_by()` is most useful for `summarise()`, but also works for `mutate()`, `filter()`, `slice()`, and `arrange()`.

What proportion are passengers and crew at each port?

```
airport_screening %>%
  group_by(arrival_port, passenger_crew) %>%
  summarise(n = n()) %>%
  group_by(arrival_port) %>%
  mutate(prop_of_port = n / sum(n)) %>%
  ungroup()
```

``summarise()`` has grouped output by 'arrival_port'. You can override using the ``.groups`` argument.

A tibble: 10 × 4

	arrival_port <chr>	passenger_crew <chr>	n <int>	prop_of_port <dbl>
1	ADL	P	8	1
2	BNE	C	3	0.111
3	BNE	P	24	0.889
4	DRW	P	5	1
5	MEL	P	61	1
6	OOL	P	5	1
7	PER	C	1	0.125
8	PER	P	7	0.875
9	SYD	C	4	0.0465
10	SYD	P	82	0.953

Which check-in port had the most trips for each arrival port?

```
airport_screening %>%
  group_by(arrival_port, check_in_port) %>%
  summarise(total_trips = sum(number_trips, na.rm = TRUE)) %>%
  group_by(arrival_port) %>%
  filter(total_trips == max(total_trips)) %>%
  ungroup()
```

``summarise()`` has grouped output by 'arrival_port'. You can override using the ``.groups`` argument.

A tibble: 7 × 3

	arrival_port <chr>	check_in_port <chr>	total_trips <dbl>
1	ADL	SIN	156
2	BNE	DFW	949
3	DRW	DPS	21
4	MEL	AKL	608
5	OOL	AKL	27
6	PER	HKG	72
7	SYD	SIN	713

Applying the same operation to multiple columns

We can use `across()` to get summaries of multiple variables at once.

The first parameter is used to select columns, or ranges of columns using `:`, the same as you would inside `select()`.

The second parameter is the operation to perform. Note the tilde/squiggle `~` before the operation; `.` is used as a placeholder for the variable to operate on.

```
pig_behaviour_data %>%
  group_by(Housing, Treatment) %>%
  summarise(across(Upright:Nosing_pen,
                    ~mean(., na.rm = TRUE))) %>%
  ungroup()
```

A tibble: 4 × 9

	Housing	Treatment	Upright	Aggression	Chewing_pig	Playing	Vocalising
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	FC	C	5.27	0.25	2.08	0.217	1.15
2	FC	HC	6.01	0.153	1.78	0.188	1.07
3	PS	C	7.24	0.143	2.3	0.257	1.41
4	PS	HC	7.23	0.340	2.38	0.113	1.66

i 2 more variables: Exploring_pen <dbl>, Nosing_pen <dbl>

This works in `mutate()` too. In this example we convert all of our categorical variables to factors at once:

```
pig_behaviour_data %>%
  mutate(across(c(Pen:Sex, Time, Time_brief),
                 ~as.factor(.))) %>%
  glimpse()
```

Rows: 280

Columns: 16

\$ Pen	<fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,...
\$ Housing	<fct> FC, FC, FC, FC, FC, FC, PS, PS, PS, PS, FC, FC...
\$ Treatment	<fct> HC, HC, HC, HC, HC, HC, HC, HC, HC, HC, HC, HC...
\$ HousTreat	<fct> "FC, HC", "FC, HC", "FC, HC", "FC, HC", "FC, H...
\$ Sex	<fct> M, M, M, M, M, M, M, M, M, M, F, F, F, F, F, F...
\$ Total_pigs	<dbl> 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10...
\$ Time	<fct> 15 mins, 15 mins, 15 mins, 15 mins, 15 mins, 1...
\$ Time_brief	<fct> 15m, 15m, 15m, 15m, 15m, 15m, 15m, 15m, 15m, 1...
\$ Time_hours	<dbl> 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25...
\$ Upright	<dbl> 10, 10, 8, 10, 10, 10, 10, 10, 10, 10, 10, 10...
\$ Aggression	<dbl> 0, 0, 1, 2, 0, 2, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0...
\$ Chewing_pig	<dbl> 2, 0, 3, 0, 0, 0, 2, 1, 0, 3, 5, 0, 1, 0, 3, 3...
\$ Playing	<dbl> 0, 1, 0, 0, 3, 4, 2, 1, 0, 0, 1, 0, 0, 0, 0, 0...
\$ Vocalising	<dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
\$ Exploring_pen	<dbl> 8, 10, 5, 8, 7, 6, 6, 8, 7, 6, 7, 9, 10, 9, 0,...
\$ Nosing_pen	<dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0...

Applying the same operation to multiple columns

You can mix `across()` with single variable summaries, and also apply multiple operations to each variable. The syntax for this is a bit cumbersome:

```
pig_behaviour_data %>%
  group_by(Housing, Treatment) %>%
  summarise(n = n(),
            across(Upright:Nosing_pen,
                  list(mean = ~mean(., na.rm = TRUE),
                      sd = ~sd(., na.rm = TRUE)))) %>%
  ungroup()
```

```
# A tibble: 4 × 17
  Housing Treatment      n Upright_mean Upright_sd Aggression_mean
  <chr>    <chr>    <int>      <dbl>      <dbl>          <dbl>
1 FC      C        64       5.27       4.08           0.25
2 FC      HC       88       6.01       3.94           0.153
3 PS      C       72       7.24       3.14           0.143
4 PS      HC       56       7.23       3.45           0.340
# i 11 more variables: Aggression_sd <dbl>, Chewing_pig_mean <dbl>,
#   Chewing_pig_sd <dbl>, Playing_mean <dbl>, Playing_sd <dbl>,
#   Vocalising_mean <dbl>, Vocalising_sd <dbl>,
#   Exploring_pen_mean <dbl>, Exploring_pen_sd <dbl>,
#   Nosing_pen_mean <dbl>, Nosing_pen_sd <dbl>
```

Occasionally we want to do the same thing to **every** column, which we can do by using `everything()`. Here we use it to count missing values.

```
pig_behaviour_data %>%
  group_by(Housing, Treatment) %>%
  summarise(across(everything(), ~sum(is.na(.)))) %>%
  ungroup() %>%
  glimpse()
```

```
Rows: 4
Columns: 16
$ Housing      <chr> "FC", "FC", "PS", "PS"
$ Treatment    <chr> "C", "HC", "C", "HC"
$ Pen          <int> 0, 0, 0, 0
$ HousTreat    <int> 0, 0, 0, 0
$ Sex          <int> 0, 0, 0, 0
$ Total_pigs   <int> 0, 0, 0, 0
$ Time         <int> 0, 0, 0, 0
$ Time_brief   <int> 0, 0, 0, 0
$ Time_hours   <int> 0, 0, 0, 0
$ Upright      <int> 4, 3, 2, 3
$ Aggression    <int> 4, 3, 2, 3
$ Chewing_pig   <int> 4, 3, 2, 3
$ Playing       <int> 4, 3, 2, 3
$ Vocalising   <int> 4, 3, 2, 3
$ Exploring_pen <int> 4, 3, 2, 3
```

Rowwise summary operations

`rowwise()` is a special kind of grouping operation that makes each row its own group.

`c_across()` makes a vector going across a sequence of columns.

Put these together to use summary operations (`mean()`, `sum()`, `sd()`, etc) across instead of down.

```
pig_behaviour_data %>%  
  rowwise() %>%  
  mutate(Activity_total = sum(c_across(Aggression:Nosing_pen))) %>%  
  ungroup() %>%  
  glimpse()
```

```
Rows: 280  
Columns: 17  
$ Pen      <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...  
$ Housing  <chr> "FC", "FC", "FC", "FC", "FC", "FC", "PS", "PS...  
$ Treatment <chr> "HC", "HC", "HC", "HC", "HC", "HC", "HC", "HC...  
$ HousTreat <chr> "FC, HC", "FC, HC", "FC, HC", "FC, HC", "FC, ...  
$ Sex      <chr> "M", "M", "M", "M", "M", "M", "M", "M", "M", ...  
$ Total_pigs <dbl> 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 1...  
$ Time     <chr> "15 mins", "15 mins", "15 mins", "15 mins", "...  
$ Time_brief <chr> "15m", "15m", "15m", "15m", "15m", "15m", "15...  
$ Time_hours <dbl> 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.2...  
$ Upright  <dbl> 10, 10, 8, 10, 10, 10, 10, 10, 10, 10, 10, 10...  
$ Aggression <dbl> 0, 0, 1, 2, 0, 2, 0, 0, 0, 2, 0, 0, 2, 0, 0, ...  
$ Chewing_pig <dbl> 2, 0, 3, 0, 0, 0, 2, 1, 0, 3, 5, 0, 1, 0, 3, ...  
$ Playing  <dbl> 0, 1, 0, 0, 3, 4, 2, 1, 0, 0, 1, 0, 0, 0, 0, ...  
$ Vocalising <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  
$ Exploring_pen <dbl> 8, 10, 5, 8, 7, 6, 6, 8, 7, 6, 7, 9, 10, 9, 0...  
$ Nosing_pen <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ...  
$ Activity_total <dbl> 10, 11, 9, 10, 10, 12, 10, 10, 8, 11, 13, 9, ...
```

Joining two tables

Sometimes you have two different datasets with information on the same experimental units. How do you combine them?

Common example: demographic data in one data frame, experimental results in another data frame.

demographic_data

```
# A tibble: 3 × 3
  participant_id gender  age
      <dbl> <chr> <dbl>
1             1 F      39
2             2 M      55
3             3 F      87
```

experiment_data

```
# A tibble: 6 × 3
  participant_id time  score
      <dbl> <chr> <dbl>
1             1 Pre    13
2             1 Post     9
3             2 Pre    15
4             2 Post    14
5             3 Pre     9
6             3 Post    11
```

```
left_join(experiment_data,
          demographic_data,
          by = "participant_id")
```

```
# A tibble: 6 × 5
  participant_id time  score gender  age
      <dbl> <chr> <dbl> <chr> <dbl>
1             1 Pre    13 F      39
2             1 Post     9 F      39
3             2 Pre    15 M      55
4             2 Post    14 M      55
5             3 Pre     9 F      87
6             3 Post    11 F      87
```

`left_join()` keeps all rows in the first data frame and adds in matching information on the second data frame (or `NA` if there is no match).

Other useful join functions: `right_join()`, `inner_join()`, `full_join()`

Appending vectors and data frames

You can also use `c()` to combine two (or more) vectors, one after another.

```
c(melbourne_postcodes$suburb_name,  
  hipster_postcodes$suburb_name)
```

```
[1] "Melbourne"      "East Melbourne"  "West Melbourne"  
[4] "North Melbourne" "South Melbourne" "Brunswick West"  
[7] "Brunswick"      "Brunswick East"  "Northcote"
```

If you have two (or more) data frames with the same columns, you can stack them using `bind_rows()`.

```
bind_rows(melbourne_postcodes, hipster_postcodes)
```

```
# A tibble: 9 × 2  
  postcode suburb_name  
    <dbl> <chr>  
1    3000 Melbourne  
2    3002 East Melbourne  
3    3003 West Melbourne  
4    3051 North Melbourne  
5    3205 South Melbourne  
6    3055 Brunswick West  
7    3056 Brunswick  
8    3057 Brunswick East  
9    3070 Northcote
```

Tidy data

Recall our advice for organising your data:

- Every row represents an observation of an experimental unit.
- Every column represents a variable.

Often, we find out that what we (or somebody else) thought was a variable was really a combination of two variables.

Most common example: **time**. For example, baseline and post-experiment values may be stored in different columns. It is often better to have a column named `time` and a different row for each time point.

Other common examples: location, experimental condition.

Slightly unusual example: the 7 types of pig behaviour.

Converting from wide to long form

Some made-up data, perhaps typical of a real study where before and after measurements are made on participants:

```
experiment_data <- tribble(
  ~id, ~pre, ~post,
  1,   13,   9,
  2,   15,  14,
  3,   9,   11
)
```

For the purposes of visualisation or analysis, we may want this in long form, with the pre and post measurements as separate observations.

```
experiment_long <- experiment_data %>%
  pivot_longer(c(pre, post),
               names_to = "time",
               values_to = "score")
experiment_long
```

```
# A tibble: 6 × 3
   id time  score
<dbl> <chr> <dbl>
1     1 pre     13
2     1 post      9
3     2 pre     15
4     2 post     14
5     3 pre      9
6     3 post     11
```

Pivoting multiple outcomes

```
experiment2_data <- tribble(
  ~id, ~english_pre, ~english_post, ~maths_pre, ~maths_post,
  1,    13,          9,            15,        17,
  2,    15,          14,           9,         10,
  3,    9,           11,          14,         10
)
```

Now we have a more complex scenario. Each variable name contains two pieces of information, the subject (`english` or `maths`) and the time (`pre` or `post`). We need to provide `names_sep = "_"` so `pivot_longer()` knows how to distinguish the two parts of the variable name.

```
experiment2_long <- experiment2_data %>%
  pivot_longer(english_pre:maths_post,
               names_to = c("subject", "time"),
               names_sep = "_",
               values_to = "score")
experiment2_long
```

```
# A tibble: 12 × 4
      id subject time  score
  <dbl> <chr>   <chr> <dbl>
1     1 english pre     13
2     1 english post     9
3     1 maths  pre     15
4     1 maths  post    17
5     2 english pre     15
6     2 english post    14
7     2 maths  pre      9
8     2 maths  post    10
9     3 english pre      9
10    3 english post    11
11    3 maths  pre     14
12    3 maths  post    10
```

Pivoting multiple outcomes

```
experiment2_data <- tribble(
  ~id, ~english_pre, ~english_post, ~maths_pre, ~maths_post,
  1, 13, 9, 15, 17,
  2, 15, 14, 9, 10,
  3, 9, 11, 14, 10
)
```

Often the previous data was *too long* for what we want to do.

The placeholder `.value` indicates that part of the original column name should be kept. We no longer need to provide `values_to`.

```
experiment2_long <- experiment2_data %>%
  pivot_longer(english_pre:maths_post,
               names_to = c(".value", "time"),
               names_sep = "_")
experiment2_long
```

A tibble: 6 × 4

	id	time	english	maths
	<dbl>	<chr>	<dbl>	<dbl>
1	1	pre	13	15
2	1	post	9	17
3	2	pre	15	9
4	2	post	14	10
5	3	pre	9	14
6	3	post	11	10

Converting from long to wide form

Transforming the data into a wider form is less commonly needed for analytic purposes but is sometimes helpful when making human-friendly tables. In this example, we reverse the transformation we did before and end up with the original data again.

```
experiment2_long
```

```
# A tibble: 6 × 4
```

	id	time	english	maths
	<dbl>	<chr>	<dbl>	<dbl>
1	1	pre	13	15
2	1	post	9	17
3	2	pre	15	9
4	2	post	14	10
5	3	pre	9	14
6	3	post	11	10

```
experiment2_long %>%  
  pivot_wider(names_from = time,  
              values_from = c(maths, english))
```

```
# A tibble: 3 × 5
```

	id	maths_pre	maths_post	english_pre	english_post
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	1	15	17	13	9
2	2	9	10	15	14
3	3	14	10	9	11

Exercise 3.3.