Recoding, replacing, and looking up

Packages and data

```
library(tidyverse)
my_url <- "http://datafiles.ritsokiguess.site/ais.txt"
athletes <- read_tsv(my_url)
```

Recoding

- When you want to create a new column based on an old column, in a way beyond a simple formula, you are "recoding"
- If you recode a categorical variable, `recode_values` will do the job for you.
- Make numeric codes: female = 1, male = 2:

```
athletes %>%
  mutate(sex_code = recode_values(
    Sex,
    "female" ~ 1,
    "male"   ~ 2
  )) %>%
  select(Sex, sex_code) -> d
```

The result

```
d %>% slice_sample(n = 10)
```

```
# A tibble: 10 x 2
   Sex    sex_code
   <chr>     <dbl>
 1 female        1
 2 male          2
 3 male          2
 4 female        1
 5 male          2
 6 female        1
 7 female        1
 8 male          2
 9 female        1
10 male          2
```

Recoding another categorical variable

Categorize the sports by where they are played: a court, as part of track and field, on or in water:

```
athletes %>%
  mutate(where_played = recode_values(
    Sport,
    c("BBall", "Netball", "Tennis") ~ "court",
    c("Row", "Swim", "WPolo")       ~ "water",
    c("Field", "T400m", "TSprnt")   ~ "track&field",
    default                         = Sport
  )) %>%
  select(Sport, where_played) -> d
```

The result

```
d %>% slice_sample(n = 10)
```

```
# A tibble: 10 x 2
   Sport   where_played
   <chr>   <chr>
 1 Netball court
 2 Row     water
 3 Field   track&field
 4 Swim    water
 5 Swim    water
 6 Row     water
 7 Row     water
 8 BBall   court
 9 Netball court
10 TSprnt  track&field
```

Recoding a quantitative variable 1/2

for example, creating a new column categorizing the RCC value as low or high, use `case_when`:

```
athletes %>%
  mutate(rcc_status = case_when(
    RCC < 4.5  ~ "rcc_low",
    RCC >= 4.5 ~ "rcc_high"
  )) %>%
  select(RCC, rcc_status) -> d
```

Result 1

```
d %>% slice_sample(n = 10)
```

```
# A tibble: 10 x 2
     RCC rcc_status
   <dbl> <chr>
 1  5.69 rcc_high
 2  5.11 rcc_high
 3  3.95 rcc_low
 4  4.57 rcc_high
 5  4.45 rcc_low
 6  3.95 rcc_low
 7  4.11 rcc_low
 8  4.53 rcc_high
 9  4.2  rcc_low
10  4.51 rcc_high
```

Recoding a quantitative variable 2/2

There is also `between`, if we instead wished to do this:

```
athletes %>%
  mutate(rcc_status = case_when(
    RCC < 4.0                 ~ "rcc_low",
    between(RCC, 4.0, 4.5) ~ "rcc_medium",
    RCC > 4.5                 ~ "rcc_high"
  )) %>%
  select(RCC, rcc_status) -> d
```

Result 2

```
d %>% slice_sample(n = 10)
```

```
# A tibble: 10 x 2
     RCC rcc_status
   <dbl> <chr>
 1  4.63 rcc_high
 2  5.02 rcc_high
 3  4.87 rcc_high
 4  4.92 rcc_high
 5  4.78 rcc_high
 6  4.32 rcc_medium
 7  4.87 rcc_high
 8  5.4  rcc_high
 9  4.51 rcc_high
10  4.3  rcc_medium
```

Replacing a few values in a column

- To make a few small changes to a column (rather than create a whole new column):
  - `replace_values` (like `recode_values`)
  - `replace_when` (like `case_when`).
- These change a few values and leave everything else unchanged.

Replacing a few values in a categorical column

For the athletes, let's change the two track running events to "Track":

```
athletes %>%
  mutate(Sport = replace_values(
    Sport,
    c("T400m", "TSprnt") ~ "Track"
    )) -> d
```

The result

```
d %>% slice_sample(n = 10)
```

```
# A tibble: 10 x 13
   Sex    Sport   RCC   WCC    Hc    Hg  Ferr   BMI   SSF `%Bf
   <chr>  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <d
 1 female Row    4.41   5.9  41.1  13.5    41  24.0 124.    22
 2 male   Track  5.69  10.8  50.5  18.5    53  24.5  42.3    7
 3 male   Row    4.71   8    45.5  15.6    91  24.9  78      12
 4 male   Row    5.4    6.8  49.5  17.3   183  26.1  44.7     8
 5 male   BBall  4.87   7.4  43.5  15      49  22.4  43.8     7
 6 male   Field  4.94   6.3  45.7  15.5    50  23.1  34.3     6
 7 male   Swim   4.83   7.6  45.2  15.2    97  23.9  41.8     7
 8 female Field  4.58   5.8  42.1  14.7   164  28.6 110.    21
 9 male   Track  4.99   7.2  41.4  14.9    44  22.4  36.6     5
10 male   Track  5.34   7.6  48.3  16.2    91  21.0  44       7
# i 1 more variable: Wt <dbl>
```

Making small changes in a quantitative column

Let's suppose we know that an `RCC` value less than 4 cannot be correct, and we want to replace all such values by `NA` ("missing"). Suppose the highest possible value is 4.5, and we also want to replace values higher than that by 4.5. This is what `replace_when` is for:

```
athletes %>%
  mutate(RCC = replace_when(
    RCC,
    RCC < 4   ~ NA,
    RCC > 4.5 ~ 4.5
  )) -> d
```

The result

```
d %>% slice_sample(n = 10)
```

```
# A tibble: 10 x 13
   Sex    Sport   RCC   WCC    Hc    Hg  Ferr   BMI   SSF `%Bf
   <chr>  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <d
 1 female Row    NA     3.3  36.9  12.5    40  24.5  74.9    16
 2 female Gym     4.19  9    39    13.4    69  18.9  43.5    10
 3 male   BBall  4.5   7.5  41.6  14.4    53  23.7  48.3     8
 4 female BBall  4.3   8.9  41.1  13.5    41  22.6  75.1    18
 5 male   T400m  4.5   7.5  47.5  16.5    20  21.9  46.7     9
 6 male   Row    4.5   8.2  43.8  15     130  23.6  49.2     9
 7 male   Row    4.5   7.1  44    14.8    64  25.8  61.8    12
 8 female Row    4.41  5.9  41.1  13.5    41  24.0 124.     22
 9 male   WPolo  4.5  10    46.8  16.2    94  25.8 101.     17
10 male   T400m  4.5   6.6  43.8  14.3    53  21.7  30.9     6
# i 1 more variable: Wt <dbl>
```

Making small changes to columns using a lookup table

Example: 12 cities and states, but some of the states are wrong (should not be Wisconsin):

```
my_url <- "https://datafiles.ritsokiguess.site/wisc_wrong.csv"
wisc <- read_csv(my_url)
```

The dataframe

```
wisc
```

```
# A tibble: 12 x 2
   location      state
   <chr>         <chr>
 1 Appleton      WI
 2 Beloit        WI
 3 Fort.Atkinson WI
 4 Madison       WI
 5 Marshfield    WI
 6 Milwaukee     WI
 7 Monroe        WI
 8 Superior      WI
 9 Wausau        WI
10 Dubuque       WI
11 St.Paul       WI
12 Chicago       WI
```

Setting up corrections

Some of these cities are in the wrong state: Dubuque is in Iowa (IA), St. Paul in Minnesota (MN), and Chicago is in Illinois (IL). First make a "lookup table" with the corrections we want to make:

```
corrections <- tribble(
  ~location, ~state,
  "Dubuque", "IA",
  "St.Paul", "MN",
  "Chicago", "IL"
)
corrections
```

```
# A tibble: 3 x 2
  location state
  <chr>    <chr>
1 Dubuque  IA
2 St.Paul  MN
3 Chicago  IL
```

Note: columns of this dataframe have *same names* as the ones in `wisc`.

Applying the corrections

```
wisc %>%
  rows_update(corrections, by = "location")
```

```
# A tibble: 12 x 2
   location      state
   <chr>         <chr>
 1 Appleton      WI
 2 Beloit        WI
 3 Fort.Atkinson WI
 4 Madison       WI
 5 Marshfield    WI
 6 Milwaukee     WI
 7 Monroe        WI
 8 Superior      WI
 9 Wausau        WI
10 Dubuque       IA
11 St.Paul       MN
12 Chicago       IL
```

Addendum: does this work with `replace_when` ?

Yes, but you have to specify the corrections by hand:

```
wisc %>%
  mutate(state = replace_when(
    state,
    location == "Dubuque" ~ "IA",
    location == "St.Paul" ~ "MN",
    location == "Chicago" ~ "IL"
  )) -> d
```

The result

```
d
```

```
# A tibble: 12 x 2
   location      state
   <chr>         <chr>
 1 Appleton      WI
 2 Beloit        WI
 3 Fort.Atkinson WI
 4 Madison       WI
 5 Marshfield    WI
 6 Milwaukee     WI
 7 Monroe        WI
 8 Superior      WI
 9 Wausau        WI
10 Dubuque       IA
11 St.Paul       MN
12 Chicago       IL
```

Looking things up in another data frame

- Suppose you are working in the nails department of a hardware store and you find that you have sold these items:

```
my_url <- "http://ritsokiguess.site/datafiles/nail_sales.csv"
sales <- read_csv(my_url)
sales
```

```
# A tibble: 6 x 2
  product_code sales
  <chr>        <dbl>
1 061-5344-6      10
2 161-0090-0       6
3 061-5388-2       2
4 161-0199-4       8
5 061-5375-2       5
6 061-4525-2       3
```

Product descriptions and prices

- but you don't remember what these product codes are, and you would like to know the total revenue from these sales.
- Fortunately you found a list of product descriptions and prices:

```
my_url <- "http://ritsokiguess.site/datafiles/nail_desc.csv"
desc <- read_csv(my_url)
desc
```

```
# A tibble: 7 x 5
  product_code description    size        qty price
  <chr>        <chr>          <chr>     <dbl> <dbl>
1 061-4525-2   spike nail     "10\""        1  1.49
2 061-5329-4   masonry nail   "1.5\""     112  8.19
3 061-5344-6   finishing nail "1\""       1298  6.99
4 061-5375-2   roofing nail   "1.25\""    192  6.99
5 061-5388-2   framing nail   "4\""        25  8.19
6 161-0090-0   wood nail      "1\""        25  2.39
7 161-0199-4   panel nail     "1-5/8\""    20  4.69
```

The lookup

- How do you "look up" the product codes to find the product descriptions and prices?
- `left_join`.

```
sales %>% left_join(desc)
```

```
# A tibble: 6 x 6
  product_code sales description    size         qty price
  <chr>        <dbl> <chr>          <chr>      <dbl> <dbl>
1 061-5344-6      10 finishing nail "1\""       1298  6.99
2 161-0090-0       6 wood nail      "1\""         25  2.39
3 061-5388-2       2 framing nail   "4\""         25  8.19
4 161-0199-4       8 panel nail     "1-5/8\""     20  4.69
5 061-5375-2       5 roofing nail   "1.25\""     192  6.99
6 061-4525-2       3 spike nail     "10\""         1  1.49
```

What we have

- this looks up all the rows in the *first* dataframe that are also in the *second*.
- by default matches all columns with same name in two dataframes (`product_code` here)
- get *all* columns in *both* dataframes. The rows are the ones for that `product_code`.

So now can work out how much the total revenue was:

```
sales %>% left_join(desc) %>%
  mutate(product_revenue = sales*price) %>%
  summarize(total_revenue = sum(product_revenue))
```

```
# A tibble: 1 x 1
  total_revenue
          <dbl>
1          178.
```

More comments

- if any product codes are not matched, you get NA in the added columns
- anything in the *second* dataframe that was not in the first does not appear (here, any products that were not sold)
- other variations (examples follow):
  - if there are two columns with the same name in the two dataframes, and you only want to match on one, use `by` with one column name
  - if the columns you want to look up have different names in the two dataframes, use `by` with a "named list"

Matching on only some matching names

- Suppose the `sales` dataframe *also* had a column `qty` (which was the quantity sold):

```
sales %>% rename("qty"="sales") -> sales1
sales1
```

```
# A tibble: 6 x 2
  product_code   qty
  <chr>        <dbl>
1 061-5344-6      10
2 161-0090-0       6
3 061-5388-2       2
4 161-0199-4       8
5 061-5375-2       5
6 061-4525-2       3
```

- The `qty` in `sales1` is the quantity sold, but the `qty` in `desc` is the number of nails in a package. These should *not* be matched: they are different things.

Matching only on product code

```
sales1 %>%
  left_join(desc, join_by(product_code))
```

```
# A tibble: 6 x 6
  product_code qty.x description    size       qty.y price
  <chr>         <dbl> <chr>          <chr>      <dbl> <dbl>
1 061-5344-6       10 finishing nail "1\""       1298  6.99
2 161-0090-0        6 wood nail      "1\""         25  2.39
3 061-5388-2        2 framing nail   "4\""         25  8.19
4 161-0199-4        8 panel nail     "1-5/8\""     20  4.69
5 061-5375-2        5 roofing nail   "1.25\""     192  6.99
6 061-4525-2        3 spike nail     "10\""         1  1.49
```

- Get `qty.x` (from `sales1`) and `qty.y` (from `desc`).

Matching on different names 1/2

- Suppose the product code in `sales` was just `code`:

```
sales %>% rename("code" = "product_code") -> sales2
sales2
```

```
# A tibble: 6 x 2
  code        sales
  <chr>       <dbl>
1 061-5344-6     10
2 161-0090-0      6
3 061-5388-2      2
4 161-0199-4      8
5 061-5375-2      5
6 061-4525-2      3
```

- How to match the two product codes that have different names?

Matching on different names 2/2

- Use `join_by`, but like this:

```
sales2 %>%
  left_join(desc, join_by(code == product_code))
```

```
# A tibble: 6 x 6
  code        sales description    size        qty price
  <chr>       <dbl> <chr>          <chr>     <dbl> <dbl>
1 061-5344-6     10 finishing nail "1\""      1298  6.99
2 161-0090-0      6 wood nail      "1\""        25  2.39
3 061-5388-2      2 framing nail   "4\""        25  8.19
4 161-0199-4      8 panel nail     "1-5/8\""    20  4.69
5 061-5375-2      5 roofing nail   "1.25\""    192  6.99
6 061-4525-2      3 spike nail     "10\""        1  1.49
```

Other types of join

- `right_join`: interchanges roles, looking up keys from second dataframe in first.
- `anti_join`: give me all the rows in the first dataframe that are *not* in the second. (Use this eg. to see whether the product descriptions are incomplete.)
- `full_join`: give me all the rows in both dataframes, with missings as needed.

Full join here

```
sales %>% full_join(desc)
```

```
# A tibble: 7 x 6
  product_code sales description    size         qty price
  <chr>        <dbl> <chr>          <chr>      <dbl> <dbl>
1 061-5344-6      10 finishing nail "1\""       1298  6.99
2 161-0090-0       6 wood nail      "1\""         25  2.39
3 061-5388-2       2 framing nail   "4\""         25  8.19
4 161-0199-4       8 panel nail     "1-5/8\""     20  4.69
5 061-5375-2       5 roofing nail   "1.25\""     192  6.99
6 061-4525-2       3 spike nail     "10\""         1  1.49
7 061-5329-4      NA masonry nail   "1.5\""      112  8.19
```

- The missing `sales` for "masonry nail" says that it was in the lookup table `desc`, but we didn't sell any.

The same thing, but with anti_join

Anything in first df but not in second?

```
desc %>% anti_join(sales)
```

```
# A tibble: 1 x 5
  product_code description  size       qty price
  <chr>        <chr>        <chr>    <dbl> <dbl>
1 061-5329-4   masonry nail "1.5\""    112  8.19
```

Masonry nails are the only thing in our product description file that we did not sell any of.

The other way around

```
sales %>% anti_join(desc)
```

```
# A tibble: 0 x 2
# i 2 variables: product_code <chr>, sales <dbl>
```

There was nothing we sold that was not in the description file.