Grouped data

dplyr verbs are particularly powerful when you apply them to grouped data frames (`grouped_df` objects). This vignette shows you:

- How to group, inspect, and ungroup with `group_by()` and friends.
- How individual dplyr verbs changes their behaviour when applied to grouped data frame.
- How to access data about the "current" group from within a verb.

We'll start by loading dplyr:

```
library(dplyr)
```

group_by()

The most important grouping verb is `group_by()`: it takes a data frame and one or more variables to group by:

```
by_species <- starwars %>% group_by(species)
by_sex_gender <- starwars %>% group_by(sex, gender)
```

You can see the grouping when you print the data:

```
by_species
#> # A tibble: 87 × 14
#> # Groups:   species [38]
#>   name      height  mass hair_color skin_color eye_color birth_year sex   gender
#>   <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 Luke Sky…    172    77 blond      fair       blue            19   male  mascu…
#> 2 C-3PO        167    75 <NA>       gold       yellow         112   none  mascu…
#> 3 R2-D2         96    32 <NA>       white, bl… red             33   none  mascu…
#> 4 Darth Va…    202   136 none       white      yellow          41.9 male  mascu…
#> # ℹ 83 more rows
#> # ℹ 5 more variables: homeworld <chr>, species <chr>, films <list>,
#> #   vehicles <list>, starships <list>
by_sex_gender
#> # A tibble: 87 × 14
#> # Groups:   sex, gender [6]
#>   name      height  mass hair_color skin_color eye_color birth_year sex   gender
#>   <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 Luke Sky…    172    77 blond      fair       blue            19   male  mascu…
#> 2 C-3PO        167    75 <NA>       gold       yellow         112   none  mascu…
#> 3 R2-D2         96    32 <NA>       white, bl… red             33   none  mascu…
#> 4 Darth Va…    202   136 none       white      yellow          41.9 male  mascu…
#> # ℹ 83 more rows
#> # ℹ 5 more variables: homeworld <chr>, species <chr>, films <list>,
#> #   vehicles <list>, starships <list>
```

Or use `tally()` to count the number of rows in each group. The `sort` argument is useful if you want to see the largest groups up front.

```
by_species %>% tally()
#> # A tibble: 38 × 2
#>   species      n
#>   <chr>    <int>
#> 1 Aleena       1
#> 2 Besalisk     1
#> 3 Cerean       1
#> 4 Chagrian     1
#> # ℹ 34 more rows

by_sex_gender %>% tally(sort = TRUE)
#> # A tibble: 6 × 3
#> # Groups:   sex [5]
#>   sex    gender        n
#>   <chr>  <chr>     <int>
#> 1 male   masculine    60
#> 2 female feminine     16
#> 3 none   masculine     5
#> 4 <NA>   <NA>          4
#> # ℹ 2 more rows
```

As well as grouping by existing variables, you can group by any function of existing variables. This is equivalent to performing a `mutate()` **before** the `group_by()`:

```
bmi_breaks <- c(0, 18.5, 25, 30, Inf)

starwars %>%
  group_by(bmi_cat = cut(mass/(height/100)^2, breaks=bmi_breaks)) %>%
  tally()
#> # A tibble: 5 × 2
#>   bmi_cat       n
#>   <fct>     <int>
#> 1 (0,18.5]     10
#> 2 (18.5,25]    24
#> 3 (25,30]      13
#> 4 (30,Inf]     12
#> # ℹ 1 more row
```

Group metadata

You can see underlying group data with `group_keys()`. It has one row for each group and one column for each grouping variable:

```
by_species %>% group_keys()
#> # A tibble: 38 × 1
#>   species
```

```
#>   <chr>
#> 1 Aleena
#> 2 Besalisk
#> 3 Cerean
#> 4 Chagrian
#> # ℹ 34 more rows

by_sex_gender %>% group_keys()
#> # A tibble: 6 × 2
#>   sex            gender
#>   <chr>          <chr>
#> 1 female         feminine
#> 2 hermaphroditic masculine
#> 3 male           masculine
#> 4 none           feminine
#> # ℹ 2 more rows
```

You can see which group each row belongs to with `group_indices()`:

```
by_species %>% group_indices()
#>  [1] 11  6  6 11 11 11 11  6 11 11 11 11 34 11 24 12 11 38 36 11 11  6 31 11 11
#> [26] 18 11 11  8 26 11 21 11 11 10 10 10 11 30  7 11 11 37 32 32  1 33 35 29 11
#> [51]  3 20 37 27 13 23 16  4 38 38 11  9 17 17 11 11 11 11  5  2 15 15 11  6 25
#> [76] 19 28 14 34 11 38 22 11 11 11  6 11
```

And which rows each group contains with `group_rows()`:

```
by_species %>% group_rows() %>% head()
#> <list_of<integer>[6]>
#> [[1]]
#> [1] 46
#>
#> [[2]]
#> [1] 70
#>
#> [[3]]
#> [1] 51
#>
#> [[4]]
#> [1] 58
#>
#> [[5]]
#> [1] 69
#>
#> [[6]]
#> [1]  2  3  8 22 74 86
```

Use `group_vars()` if you just want the names of the grouping variables:

```
by_species %>% group_vars()
#> [1] "species"
by_sex_gender %>% group_vars()
#> [1] "sex"    "gender"
```

Changing and adding to grouping variables

If you apply `group_by()` to an already grouped dataset, will overwrite the existing grouping variables. For example, the following code groups by `homeworld` instead of `species`:

```
by_species %>%
  group_by(homeworld) %>%
  tally()
#> # A tibble: 49 × 2
#>   homeworld       n
#>   <chr>       <int>
#> 1 Alderaan        3
#> 2 Aleen Minor     1
#> 3 Bespin          1
#> 4 Bestine IV      1
#> # ℹ 45 more rows
```

To **augment** the grouping, using `.add = TRUE`[1]. For example, the following code groups by species and homeworld:

```
by_species %>%
  group_by(homeworld, .add = TRUE) %>%
  tally()
#> # A tibble: 57 × 3
#> # Groups:   species [38]
#>   species  homeworld       n
#>   <chr>    <chr>       <int>
#> 1 Aleena   Aleen Minor     1
#> 2 Besalisk Ojom            1
#> 3 Cerean   Cerea           1
#> 4 Chagrian Champala        1
#> # ℹ 53 more rows
```

Removing grouping variables

To remove all grouping variables, use `ungroup()`:

```
by_species %>%
  ungroup() %>%
  tally()
#> # A tibble: 1 × 1
#>       n
```

```
#>   <int>
#> 1    87
```

You can also choose to selectively ungroup by listing the variables you want to remove:

```
by_sex_gender %>%
  ungroup(sex) %>%
  tally()
#> # A tibble: 3 × 2
#>   gender        n
#>   <chr>     <int>
#> 1 feminine     17
#> 2 masculine    66
#> 3 <NA>          4
```

Verbs

The following sections describe how grouping affects the main dplyr verbs.

summarise()

`summarise()` computes a summary for each group. This means that it starts from `group_keys()`, adding summary variables to the right hand side:

```
by_species %>%
  summarise(
    n = n(),
    height = mean(height, na.rm = TRUE)
  )
#> # A tibble: 38 × 3
#>   species      n height
#>   <chr>    <int>  <dbl>
#> 1 Aleena       1     79
#> 2 Besalisk     1    198
#> 3 Cerean       1    198
#> 4 Chagrian     1    196
#> # ℹ 34 more rows
```

The `.groups=` argument controls the grouping structure of the output. The historical behaviour of removing the right hand side grouping variable corresponds to `.groups = "drop_last"` without a message or `.groups = NULL` with a message (the default).

```
by_sex_gender %>%
  summarise(n = n()) %>%
  group_vars()
#> `summarise()` has grouped output by 'sex'. You can override using the `.groups`
#> argument.
#> [1] "sex"
```

```
by_sex_gender %>%
  summarise(n = n(), .groups = "drop_last") %>%
  group_vars()
#> [1] "sex"
```

Since version 1.0.0 the groups may also be kept (`.groups = "keep"`) or dropped (`.groups = "drop"`).

```
by_sex_gender %>%
  summarise(n = n(), .groups = "keep") %>%
  group_vars()
#> [1] "sex"    "gender"

by_sex_gender %>%
  summarise(n = n(), .groups = "drop") %>%
  group_vars()
#> character(0)
```

When the output no longer have grouping variables, it becomes ungrouped (i.e. a regular tibble).

`select()`, `rename()`, and `relocate()`

`rename()` and `relocate()` behave identically with grouped and ungrouped data because they only affect the name or position of existing columns. Grouped `select()` is almost identical to ungrouped select, except that it always includes the grouping variables:

```
by_species %>% select(mass)
#> Adding missing grouping variables: `species`
#> # A tibble: 87 × 2
#> # Groups:   species [38]
#>   species  mass
#>   <chr>   <dbl>
#> 1 Human      77
#> 2 Droid      75
#> 3 Droid      32
#> 4 Human     136
#> # ℹ 83 more rows
```

If you don't want the grouping variables, you'll have to first `ungroup()`. (This design is possibly a mistake, but we're stuck with it for now.)

`arrange()`

Grouped `arrange()` is the same as ungrouped `arrange()`, unless you set `.by_group = TRUE`, in which case it will order first by the grouping variables.

```
by_species %>%
  arrange(desc(mass)) %>%
  relocate(species, mass)
```

```
#> # A tibble: 87 × 14
#> # Groups:   species [38]
#>   species  mass name     height hair_color skin_color eye_color birth_year sex
#>   <chr>   <dbl> <chr>     <int> <chr>      <chr>      <chr>          <dbl> <chr>
#> 1 Hutt     1358 Jabba D…    175 <NA>       green-tan… orange         600   herm…
#> 2 Kaleesh   159 Grievous    216 none       brown, wh… green, y…       NA   male
#> 3 Droid     140 IG-88       200 none       metal      red             15   none
#> 4 Human     136 Darth V…    202 none       white      yellow          41.9 male
#> # ℹ 83 more rows
#> # ℹ 5 more variables: gender <chr>, homeworld <chr>, films <list>,
#> #   vehicles <list>, starships <list>

by_species %>%
  arrange(desc(mass), .by_group = TRUE) %>%
  relocate(species, mass)
#> # A tibble: 87 × 14
#> # Groups:   species [38]
#>   species   mass name    height hair_color skin_color eye_color birth_year sex
#>   <chr>    <dbl> <chr>    <int> <chr>      <chr>      <chr>          <dbl> <chr>
#> 1 Aleena      15 Ratts …     79 none       grey, blue unknown           NA male
#> 2 Besalisk   102 Dexter…    198 none       brown      yellow            NA male
#> 3 Cerean      82 Ki-Adi…    198 white      pale       yellow            92 male
#> 4 Chagrian    NA Mas Am…    196 none       blue       blue              NA male
#> # ℹ 83 more rows
#> # ℹ 5 more variables: gender <chr>, homeworld <chr>, films <list>,
#> #   vehicles <list>, starships <list>
```

Note that second example is sorted by `species` (from the `group_by()` statement) and then by `mass` (within species).

mutate()

In simple cases with vectorised functions, grouped and ungrouped `mutate()` give the same results. They differ when used with summary functions:

```
# Subtract off global mean
starwars %>%
  select(name, homeworld, mass) %>%
  mutate(standard_mass = mass - mean(mass, na.rm = TRUE))
#> # A tibble: 87 × 4
#>   name           homeworld  mass standard_mass
#>   <chr>          <chr>     <dbl>         <dbl>
#> 1 Luke Skywalker Tatooine     77         -20.3
#> 2 C-3PO          Tatooine     75         -22.3
#> 3 R2-D2          Naboo        32         -65.3
#> 4 Darth Vader    Tatooine    136          38.7
#> # ℹ 83 more rows

# Subtract off homeworld mean
starwars %>%
```

```
  select(name, homeworld, mass) %>%
  group_by(homeworld) %>%
  mutate(standard_mass = mass - mean(mass, na.rm = TRUE))
#> # A tibble: 87 × 4
#> # Groups:   homeworld [49]
#>   name           homeworld  mass standard_mass
#>   <chr>          <chr>     <dbl>         <dbl>
#> 1 Luke Skywalker Tatooine     77         -8.38
#> 2 C-3PO          Tatooine     75        -10.4
#> 3 R2-D2          Naboo        32        -32.2
#> 4 Darth Vader    Tatooine    136         50.6
#> # ℹ 83 more rows
```

Or with window functions like `min_rank()`:

```
# Overall rank
starwars %>%
  select(name, homeworld, height) %>%
  mutate(rank = min_rank(height))
#> # A tibble: 87 × 4
#>   name           homeworld height  rank
#>   <chr>          <chr>      <int> <int>
#> 1 Luke Skywalker Tatooine     172    28
#> 2 C-3PO          Tatooine     167    20
#> 3 R2-D2          Naboo         96     5
#> 4 Darth Vader    Tatooine     202    72
#> # ℹ 83 more rows

# Rank per homeworld
starwars %>%
  select(name, homeworld, height) %>%
  group_by(homeworld) %>%
  mutate(rank = min_rank(height))
#> # A tibble: 87 × 4
#> # Groups:   homeworld [49]
#>   name           homeworld height  rank
#>   <chr>          <chr>      <int> <int>
#> 1 Luke Skywalker Tatooine     172     5
#> 2 C-3PO          Tatooine     167     4
#> 3 R2-D2          Naboo         96     1
#> 4 Darth Vader    Tatooine     202    10
#> # ℹ 83 more rows
```

`filter()`

A grouped `filter()` effectively does a `mutate()` to generate a logical variable, and then only keeps the rows where the variable is `TRUE`. This means that grouped filters can be used with summary functions. For example, we can find the tallest character of each species:

```
by_species %>%
  select(name, species, height) %>%
  filter(height == max(height))
#> # A tibble: 36 × 3
#> # Groups:   species [36]
#>   name                  species        height
#>   <chr>                 <chr>           <int>
#> 1 Greedo                Rodian            173
#> 2 Jabba Desilijic Tiure Hutt              175
#> 3 Yoda                  Yoda's species     66
#> 4 Bossk                 Trandoshan        190
#> # ℹ 32 more rows
```

You can also use `filter()` to remove entire groups. For example, the following code eliminates all groups that only have a single member:

```
by_species %>%
  filter(n() != 1) %>%
  tally()
#> # A tibble: 9 × 2
#>   species      n
#>   <chr>    <int>
#> 1 Droid        6
#> 2 Gungan       3
#> 3 Human       35
#> 4 Kaminoan     2
#> # ℹ 5 more rows
```

`slice()` and friends

`slice()` and friends (`slice_head()`, `slice_tail()`, `slice_sample()`, `slice_min()` and `slice_max()`) select rows within a group. For example, we can select the first observation within each species:

```
by_species %>%
  relocate(species) %>%
  slice(1)
#> # A tibble: 38 × 14
#> # Groups:   species [38]
#>   species  name    height  mass hair_color skin_color eye_color birth_year sex
#>   <chr>    <chr>    <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
#> 1 Aleena   Ratts …     79    15 none       grey, blue unknown           NA male
#> 2 Besalisk Dexter…    198   102 none       brown      yellow            NA male
#> 3 Cerean   Ki-Adi…    198    82 white      pale       yellow            92 male
#> 4 Chagrian Mas Am…    196    NA none       blue       blue              NA male
#> # ℹ 34 more rows
#> # ℹ 5 more variables: gender <chr>, homeworld <chr>, films <list>,
#> #   vehicles <list>, starships <list>
```

Similarly, we can use `slice_min()` to select the smallest `n` values of a variable:

```
by_species %>%
  filter(!is.na(height)) %>%
  slice_min(height, n = 2)
#> # A tibble: 47 × 14
#> # Groups:   species [38]
#>   name      height  mass hair_color skin_color eye_color birth_year sex   gender
#>   <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 Ratts Ty…     79    15 none       grey, blue unknown           NA male  mascu…
#> 2 Dexter J…    198   102 none       brown      yellow            NA male  mascu…
#> 3 Ki-Adi-M…    198    82 white      pale       yellow            92 male  mascu…
#> 4 Mas Amed…    196    NA none       blue       blue              NA male  mascu…
#> # ℹ 43 more rows
#> # ℹ 5 more variables: homeworld <chr>, species <chr>, films <list>,
#> #   vehicles <list>, starships <list>
```

---

1. Note that the argument changed from `add = TRUE` to `.add = TRUE` in dplyr 1.0.0.↩