What is tidy data?

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

Happy families are all alike, but every unhappy family is unhappy in its own way.

Leo Tolstoy

Tidy datasets are all alike, but every messy dataset is messy in its own way.

Hadley Wickham

Rectangular data

Structure

- Columns
- Rows
- Cells

name	homeworld	species
Luke Skywalker	Tatooine	Human
R2-D2	Naboo	Droid
Darth Vader	Tatooine	Human
Obi-Wan Kenobi	Stewjon	Human

Tidy data, variables

Structure

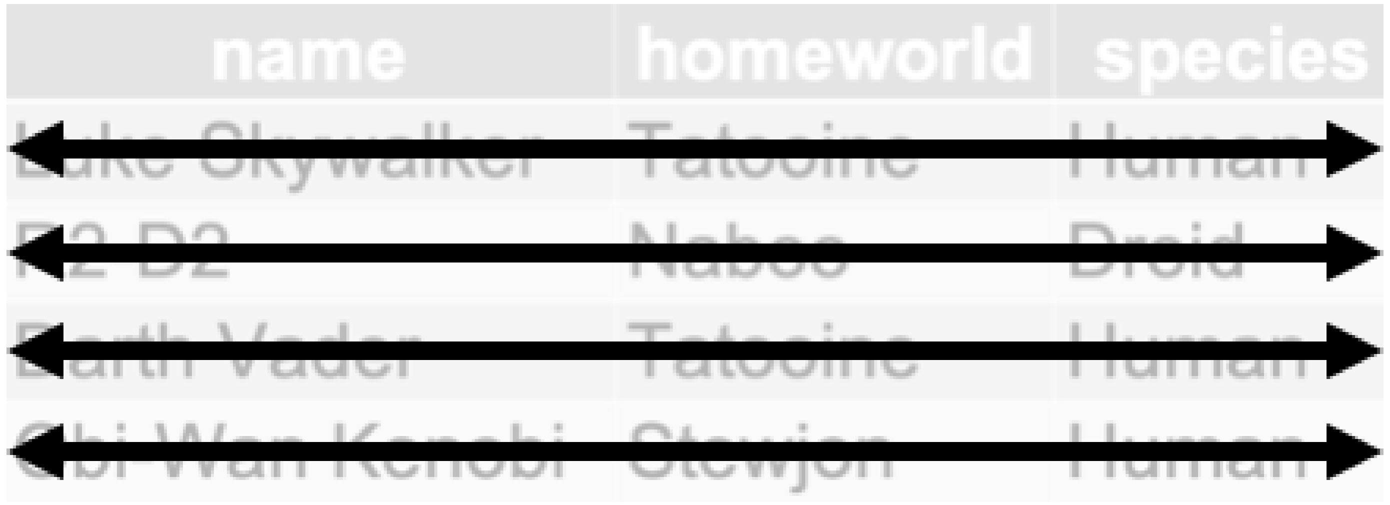
- **Columns hold variables**
- Rows
- Cells

name	homeworld	species
Luke Skywalker	Tatooine	Human
R2-D2	Naboo	Droid
Darth Vader	Tatooine	Human
Obi-Wan Kenobi	Stewjon	Human

Tidy data, observations

Structure

- Columns hold variables
- **Rows hold observations**
- Cells



Tidy data, values

Structure

- Columns hold variables
- Rows hold observations
- **Cells hold values**

name	homeworld	species
Luke Skywalker	Tatooine	Human
R2-D2	Naboo	Droid
Darth Vader	Tatooine	Human
Obi-Wan Kenobi	Stewjon	Human

dplyr recap

```
character_df
```

```
# A tibble: 4 x 3
  name           homeworld species
  <chr>          <chr>     <chr>
1 Luke Skywalker Tatooine  Human
2 R2-D2          Naboo     Droid
3 Darth Vader    Tatooine  Human
4 Obi-Wan Kenobi Stewjon   Human
```

dplyr recap: select()

```
character_df %>%
  select(name, homeworld)
```

```
# A tibble: 4 x 2
  name           homeworld
  <chr>          <chr>
1 Luke Skywalker Tatooine
2 R2-D2          Naboo
3 Darth Vader    Tatooine
4 Obi-Wan Kenobi Stewjon
```

dplyr recap: filter()

```r
character_df %>%
  filter(homeworld == "Tatooine")
```

```
# A tibble: 2 x 3
  name           homeworld species
  <chr>          <chr>     <chr>
1 Luke Skywalker Tatooine  Human
2 Darth Vader    Tatooine  Human
```

dplyr recap: mutate()

```
character_df %>%
  mutate(is_human = species == "Human")
```

```
# A tibble: 4 x 4
  name           homeworld species is_human
  <chr>          <chr>     <chr>   <lgl>
1 Luke Skywalker Tatooine  Human   TRUE
2 R2-D2          Naboo     Droid   FALSE
3 Darth Vader    Tatooine  Human   TRUE
4 Obi-Wan Kenobi Stewjon   Human   TRUE
```

dplyr recap: group_by() and summarize()

```
character_df %>%
  group_by(homeworld) %>%
  summarize(n = n())
```

```
# A tibble: 3 x 2
  homeworld     n
  <chr>     <int>
1 Naboo         1
2 Stewjon       1
3 Tatooine      2
```

[1] magrittr.tidyverse.org

[1] www.tidyverse.org

Multiple variables in a single column

```
population_df
```

```
# A tibble: 4 x 2
  country                   population
  <chr>                          <dbl>
1 Brazil, South America         210.
2 Nepal, Asia                    28.1
3 Senegal, Africa                15.8
4 Australia, Oceania             25.0
```

Separating variables over two columns

```
population_df %>%
  separate(country, into = c("country", "continent"), sep = ", ")
```

```
# A tibble: 4 x 3
  country   continent     population
  <chr>     <chr>              <dbl>
1 Brazil    South America      210.
2 Nepal     Asia                28.1
3 Senegal   Africa              15.8
4 Australia Oceania             25.0
```

Let's practice!

RESHAPING DATA WITH TIDYR

Columns with multiple values

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

Two variables in a single column

```
netflix_df
```

```
# A tibble: 637 x 3
   title                  type    duration
   <chr>                  <chr>   <chr>
 1 Article 15             Movie   125 min
 2 Kill Me If You Dare    Movie   100 min
 3 The Spy                TV Show 1 Seasons
 4 The World We Make      Movie   108 min
 5 Watchman               Movie   93 min
```

Converting separated columns' data types

```r
netflix_df %>%
  separate(duration, into = c("value", "unit"), convert = TRUE)
```

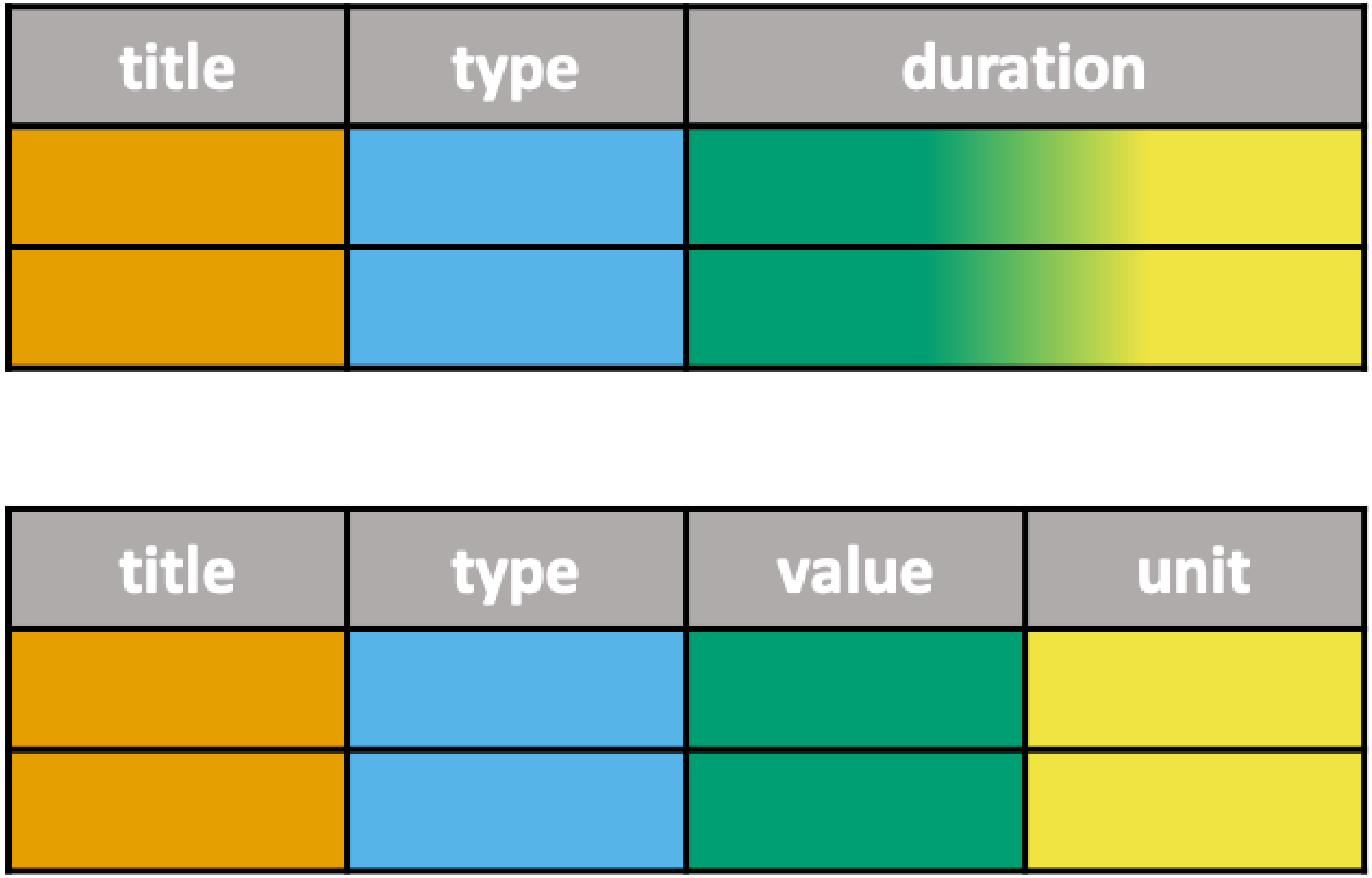
```
# A tibble: 5 x 4
  title                type    value unit
  <chr>                <chr>   <int> <chr>
1 Article 15           Movie     125 min
2 Kill Me If You Dare Movie     100 min
3 The Spy              TV Show     1 Seasons
4 The World We Make    Movie     108 min
5 Watchman             Movie      93 min
```

dplyr aggregation recap

```
netflix_df %>%
  separate(duration, into = c("value", "unit"), convert = TRUE) %>%
  group_by(type, unit) %>%
  summarize(mean_duration = mean(value))
```

```
# A tibble: 2 x 3
# Groups:   type [2]
  type    unit    mean_duration
  <chr>   <chr>           <dbl>
1 Movie   min             98.6
2 TV Show Seasons          1.85
```

Separating variables over columns



Combining multiple columns into one

```
star_wars_df
```

```
# A tibble: 4 x 2
  given_name family_name
  <chr>      <chr>
1 Luke       Skywalker
2 Han        Solo
3 Leia       Organa
4 R2         D2
```

Combining multiple columns into one

```
star_wars_df %>%
  unite("name", given_name, family_name)
```

```
# A tibble: 4 x 1
  name
  <chr>
1 Luke_Skywalker
2 Han_Solo
3 Leia_Organa
4 R2_D2
```

Combining multiple columns into one

```
star_wars_df %>%
  unite("name", given_name, family_name, sep = " ")
```

```
# A tibble: 4 x 1
  name
  <chr>
1 Luke Skywalker
2 Han Solo
3 Leia Organa
4 R2 D2
```

Multiple values in a single cell

```
drink_df
```

```
# A tibble: 2 x 2
  drink          ingredients
  <chr>          <chr>
1 Chocolate milk milk, chocolate, sugar
2 Orange juice   oranges, sugar
```

Multiple values in a single cell

Netflix data

title	type	duration

Drinks data

drink	ingredients
A	1 2 3
B	1 2

Multiple values in a single cell

Netflix data

title	type	duration

Values to variables

title	type	value	unit

Drinks data

drink	ingredients		
A	1	2	3
B	1	2	

Multiple values in a single cell

Netflix data

title	type	duration

Values to variables

title	type	value	unit

Drinks data

drink	ingredients
A	1 2 3
B	1 2

Values to observations

drink	ingredients
A	1
A	2
A	3
B	1
B	2

Separating values over rows

```
drink_df %>%
  separate_rows(ingredients, sep = ", ")
```

```
# A tibble: 5 x 2
  drink          ingredients
  <chr>          <chr>
1 Chocolate milk milk
2 Chocolate milk chocolate
3 Chocolate milk sugar
4 Orange juice   oranges
5 Orange juice   sugar
```

Counting ingredients

```r
drink_df %>%
  separate_rows(ingredients, sep = ", ") %>%
  count(drink)
```

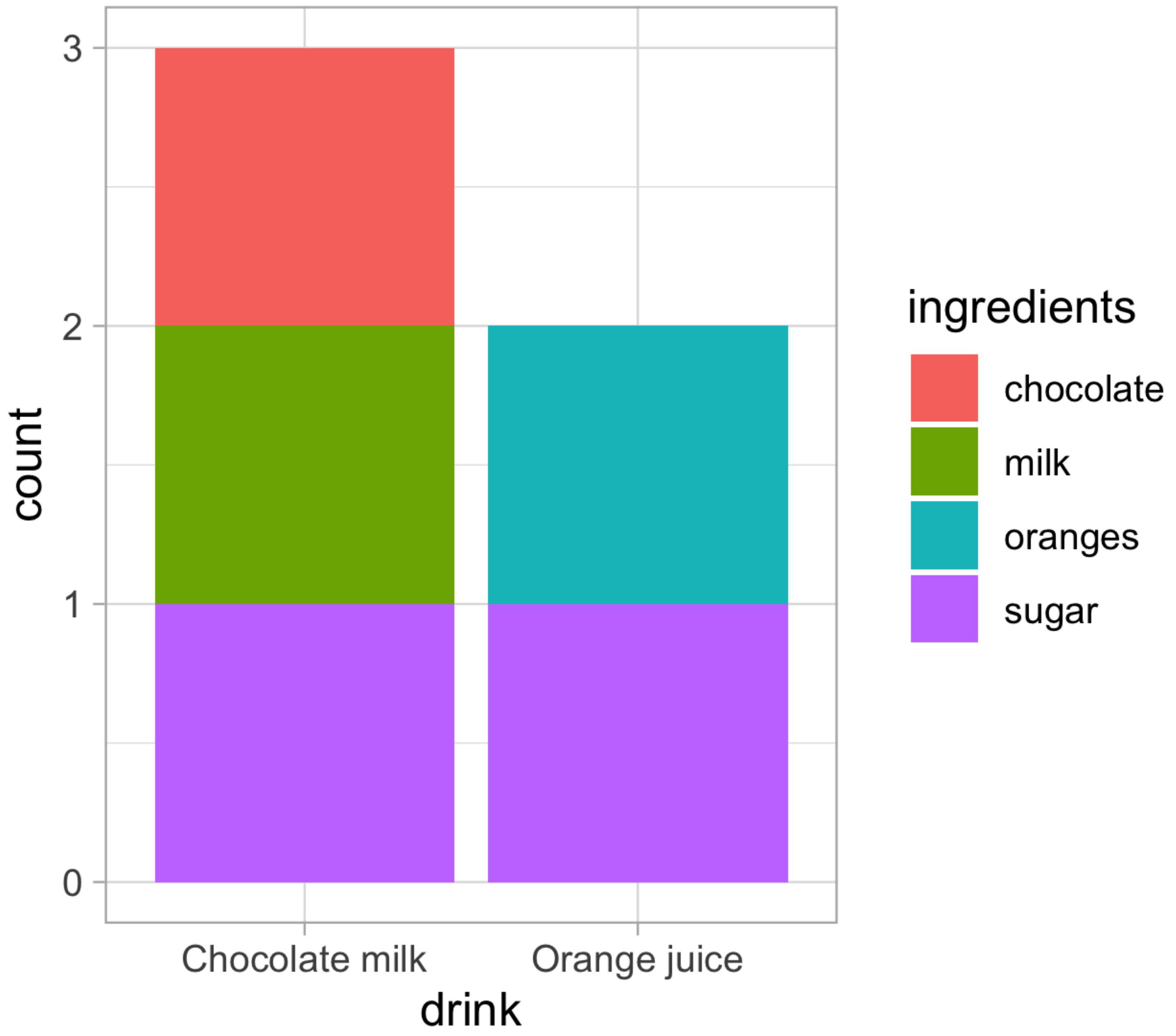
```
# A tibble: 2 x 2
  drink              n
  <chr>          <int>
1 Chocolate milk     3
2 Orange juice       2
```

```r
drink_df %>%
  separate_rows(ingredients, sep = ", ") %>%
  count(ingredients)
```

```
# A tibble: 4 x 2
  ingredients     n
  <chr>       <int>
1 chocolate       1
2 milk            1
3 oranges         1
4 sugar           2
```

Visualizing ingredients

```
drink_df %>%
  separate_rows(ingredients, sep = ", ") %>%
  ggplot(aes(x=drink, fill=ingredients)) +
  geom_bar()
```



Let's practice!

RESHAPING DATA WITH TIDYR

Missing values

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

Missing values in R

NA = Not Available

```
# A tibble: 5 x 4
  drink          ingredient quantity unit
  <chr>          <chr>         <int> <chr>
1 Chocolate milk milk              1 L
2 Chocolate milk chocolate       100 g
3 Chocolate milk sugar            20 g
4 Orange juice   oranges           3 NA
5 Orange juice   sugar            20 g
```

Imputing with a default value: replace_na()

```
moon_df
```

```
# A tibble: 4 x 2
   year people_on_moon
  <int>          <int>
1  1969              4
2  1970             NA
3  1971              4
4  1972              4
5  1973             NA
```

Imputing with a default value: replace_na()

```
moon_df %>%
  replace_na(list(people_on_moon = 0L))
```

```
# A tibble: 4 x 2
   year people_on_moon
  <int>          <int>
1  1969              4
2  1970              0
3  1971              4
4  1972              4
5  1973              0
```

```
typeof(0L)
```

```
[1] "integer"
```

```
typeof(0)
```

```
[1] "double"
```

Imputing with the most recent value: fill()

```
cumul_moon_df
```

```
# A tibble: 5 x 3
   year people_on_moon total_people_on_moon
  <int>          <int>                <int>
1  1969              4                    4
2  1970             NA                   NA
3  1971              4                    8
4  1972              4                   12
5  1973             NA                   NA
```

Imputing with the most recent value: fill()

```
cumul_moon_df %>%
  fill(total_people_on_moon)
```

```
# A tibble: 5 x 3
   year people_on_moon total_people_on_moon
  <int>          <int>                <int>
1  1969              4                    4
2  1970             NA                    4
3  1971              4                    8
4  1972              4                   12
5  1973             NA                   12
```

fill() imputation options

```
cumul_moon_df %>%
  fill(total_people_on_moon, .direction = "down")
```

```
# A tibble: 5 x 3
   year people_on_moon total_people_on_moon
  <int>          <int>                <int>
1  1969              4                    4
2  1970             NA                    4
3  1971              4                    8
4  1972              4                   12
5  1973             NA                   12
```

fill() imputation options

```
cumul_moon_df %>%
  fill(total_people_on_moon, .direction = "up")
```

```
# A tibble: 5 x 3
   year people_on_moon total_people_on_moon
  <int>          <int>                <int>
1  1969              4                    4
2  1970             NA                    8
3  1971              4                    8
4  1972              4                   12
5  1973             NA                   NA
```

Removing rows with missing values: drop_na()

```
moon_df %>%
  drop_na()
```

```
# A tibble: 3 x 2
   year people_on_moon
  <int>          <int>
1  1969              4
2  1971              4
3  1972              4
```

drop_na() caveats

```
mars_df
```

```
# A tibble: 5 x 3
   year people_on_moon people_on_mars
  <int>          <int> <int>
1  1969              4 NA
2  1970             NA NA
3  1971              4 NA
4  1972              4 NA
5  1973             NA NA
```

drop_na() caveats

```
mars_df %>%
  drop_na()
```

```
# A tibble: 0 x 3
# ... with 3 variables: year <int>, people_on_moon <int>, people_on_mars <int>
```

drop_na() caveats

```
mars_df %>%
  drop_na(people_on_moon)
```

```
# A tibble: 3 x 3
   year people_on_moon people_on_mars
  <int>          <int> <int>
1  1969              4 NA
2  1971              4 NA
3  1972              4 NA
```

Let's practice!

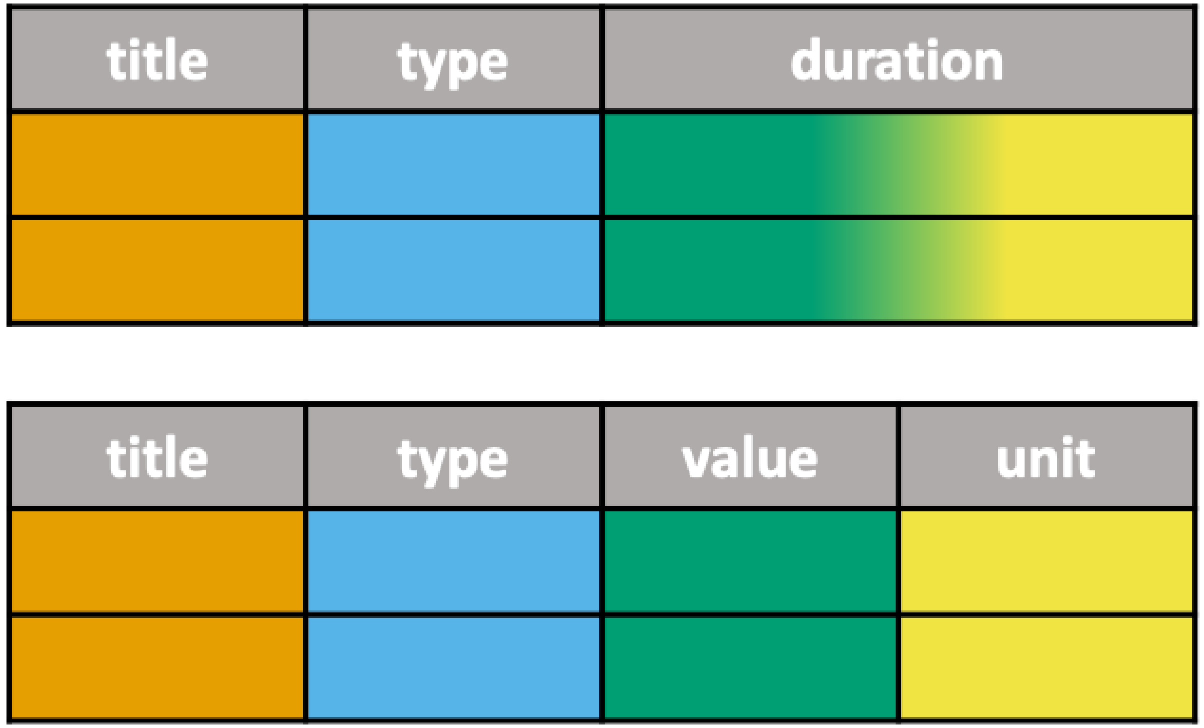
RESHAPING DATA WITH TIDYR

From wide to long data

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

`separate()`



`separate_rows()`

Values in column headers

```
nuke_df
```

```
# A tibble: 2 x 6
  country            `1945` `1946` `1948` `1949` `1951`
  <chr>               <int>  <int>  <int>  <int>  <int>
1 United States           3      2      3     NA     16
2 Russian Federation     NA     NA     NA      1      2
```

Values in column headers

country	1945	1946
USA	3	2
USSR	NA	NA

country	year	n_bombs
USA	1945	3
USA	1946	2
USSR	1945	NA
USSR	1946	NA

The pivot_longer() function

```r
nuke_df %>%
  pivot_longer(`1945`:`1951`)
```

```
# A tibble: 10 x 3
   country            name  value
   <chr>              <chr> <int>
 1 United States      1945      3
 2 United States      1946      2
 3 United States      1948      3
 4 United States      1949     NA
 5 United States      1951     16
 6 Russian Federation 1945     NA
 # ... with 4 more rows
```

The pivot_longer() function

```
nuke_df %>%
  pivot_longer(c(`1945`, `1946`, `1948`, `1949`, `1951`))
```

```
# A tibble: 10 x 3
   country            name  value
   <chr>              <chr> <int>
 1 United States      1945      3
 2 United States      1946      2
 3 United States      1948      3
 4 United States      1949     NA
 5 United States      1951     16
 6 Russian Federation 1945     NA
 # ... with 4 more rows
```

The pivot_longer() function

```
nuke_df %>%
  pivot_longer(-country)
```

```
# A tibble: 10 x 3
   country            name  value
   <chr>              <chr> <int>
 1 United States      1945      3
 2 United States      1946      2
 3 United States      1948      3
 4 United States      1949     NA
 5 United States      1951     16
 6 Russian Federation 1945     NA
 # ... with 4 more rows
```

pivot_longer() arguments

```
nuke_df %>%
  pivot_longer(-country, names_to = "year", values_to = "n_bombs")
```

```
# A tibble: 10 x 3
   country            year  n_bombs
   <chr>              <chr> <int>
 1 United States      1945      3
 2 United States      1946      2
 3 United States      1948      3
 4 United States      1949     NA
 5 United States      1951     16
 6 Russian Federation 1945     NA
 # ... with 4 more rows
```

pivot_longer() arguments

```r
nuke_df %>%
  pivot_longer(
    -country,
    names_to = "year",
    values_to = "n_bombs",
    values_drop_na = TRUE
  )
```

```
# A tibble: 6 x 3
  country            year  n_bombs
  <chr>              <chr>   <int>
1 United States      1945        3
2 United States      1946        2
3 United States      1948        3
4 United States      1951       16
5 Russian Federation 1949        1
6 Russian Federation 1951        2
```

pivot_longer() arguments

```
nuke_df %>%
  pivot_longer(
    -country,
    names_to = "year",
    values_to = "n_bombs",
    values_drop_na = TRUE,
    names_transform = list(year = as.integer)
  )
```

```
# A tibble: 6 x 3
  country             year n_bombs
  <chr>              <int>   <int>
1 United States       1945       3
2 United States       1946       2
3 United States       1948       3
4 United States       1951      16
5 Russian Federation  1949       1
6 Russian Federation  1951       2
```

Let's practice!

RESHAPING DATA WITH TIDYR

Deriving variables from column headers

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

Soviet space dogs

```
space_dogs_df
```

```
# A tibble: 42 x 4
   date       name_1   name_2   result
   <date>     <chr>    <chr>    <chr>
 1 1951-06-26 Lisa-2   Ryzhik-2 recovered safely
 2 1951-07-22 Dezik    Tsygan   recovered safely
 3 1951-07-29 Dezik    Lisa     parachute failed, both dogs died
 4 1951-08-15 Chizhik  Mishka   recovered safely
 5 1951-08-19 Ryzhik   Smeliy   recovered safely
 # ... with 37 more rows
```

Soviet space dogs: a basic pivot operation

```
dog_df %>%
  pivot_longer(
    c(name_1, name_2),
    names_to = "id",
    values_to = "name",
    values_drop_na = TRUE
  ) %>%
  select(-result)
```

```
# A tibble: 81 x 3
   date       id     name
   <date>     <chr>  <chr>
 1 1951-06-26 name_1 Lisa-2
 2 1951-06-26 name_2 Ryzhik-2
 3 1951-07-22 name_1 Dezik
 4 1951-07-22 name_2 Tsygan
 5 1951-07-29 name_1 Dezik
 6 1951-07-29 name_2 Lisa
 7 1951-08-15 name_1 Chizhik
 8 1951-08-15 name_2 Mishka
 9 1951-08-19 name_1 Ryzhik
 # ... with 72 more rows
```

Soviet space dogs: removing a prefix

```r
dog_df %>%
  pivot_longer(
    c(name_1, name_2),
    names_to = "id",
    values_to = "name",
    values_drop_na = TRUE,
    names_prefix = "name_"
  ) %>%
  select(-result)
```

```
# A tibble: 81 x 3
   date       id    name
   <date>     <chr> <chr>
 1 1951-06-26 1     Lisa-2
 2 1951-06-26 2     Ryzhik-2
 3 1951-07-22 1     Dezik
 4 1951-07-22 2     Tsygan
 5 1951-07-29 1     Dezik
 6 1951-07-29 2     Lisa
 7 1951-08-15 1     Chizhik
 8 1951-08-15 2     Mishka
 9 1951-08-19 1     Ryzhik
 # ... with 72 more rows
```

Soviet space dogs: transforming data types

```
dog_df %>%
  pivot_longer(
    c(name_1, name_2),
    names_to = "id",
    values_to = "name",
    values_drop_na = TRUE,
    names_prefix = "name_",
    names_transform = list(id = as.integer)
  ) %>%
  select(-result)
```

```
# A tibble: 81 x 3
   date          id name
   <date>     <int> <chr>
 1 1951-06-26     1 Lisa-2
 2 1951-06-26     2 Ryzhik-2
 3 1951-07-22     1 Dezik
 4 1951-07-22     2 Tsygan
 5 1951-07-29     1 Dezik
 6 1951-07-29     2 Lisa
 7 1951-08-15     1 Chizhik
 8 1951-08-15     2 Mishka
 9 1951-08-19     1 Ryzhik
 # ... with 72 more rows
```

Soviet space dogs: the starts_with() function

```r
dog_df %>%
  pivot_longer(
    starts_with("name_"),
    names_to = "id",
    values_to = "name",
    values_drop_na = TRUE,
    names_prefix = "name_",
    names_transform = list(id = as.integer)
  ) %>%
  select(-result)
```

```
# A tibble: 81 x 3
   date          id name
   <date>     <int> <chr>
 1 1951-06-26     1 Lisa-2
 2 1951-06-26     2 Ryzhik-2
 3 1951-07-22     1 Dezik
 4 1951-07-22     2 Tsygan
 5 1951-07-29     1 Dezik
 6 1951-07-29     2 Lisa
 7 1951-08-15     1 Chizhik
 8 1951-08-15     2 Mishka
 9 1951-08-19     1 Ryzhik
 # ... with 72 more rows
```

Apple revenue: two variables per column name

```
apple_revenue_df
```

```
# A tibble: 4 x 5
  segment `2019_Q1` `2019_Q2` `2019_Q3` `2019_Q4`
  <chr>       <dbl>     <dbl>     <dbl>     <dbl>
1 iPhone      52.0      31.0      26.0      33.4
2 Mac          7.42      5.51      5.82      6.99
3 iPad         6.73      4.87      5.02      4.66
4 Other       18.2      16.6      17.0      19.0
```

Apple revenue: visualizing issue and solution

segment	2019_Q1	2019_Q2
iPhone	52.0	31.0
Mac	7.42	5.51

segment	year	quarter	revenue
iPhone	2019	1	52.0
iPhone	2019	2	31.0
Mac	2019	1	7.42
Mac	2019	2	5.51

Apple revenue: Advanced pivoting

```
apple_df %>%
  pivot_longer(
    -segment,
    names_to = c("year", "quarter"),
    values_to = "revenue",
    names_sep = "_Q",
    names_transform = list(
      year = as.integer,
      quarter = as.integer
      )
  )
```

```
# A tibble: 16 x 4
   segment  year quarter revenue
   <chr>   <int>   <int>   <dbl>
 1 iPhone   2019       1   52.0
 2 iPhone   2019       2   31.0
 3 iPhone   2019       3   26.0
 4 iPhone   2019       4   33.4
 5 Mac      2019       1    7.42
 6 Mac      2019       2    5.51
 7 Mac      2019       3    5.82
 8 Mac      2019       4    6.99
 # ... with 8 more rows
```

Let's practice!

RESHAPING DATA WITH TIDYR

Deriving variables from complex column headers

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

Separating column headers into variables

segment	2019_Q1	2019_Q2
iPhone	52.0	31.0
Mac	7.42	5.51

segment	year	quarter	revenue
iPhone	2019	1	52.0
iPhone	2019	2	31.0
Mac	2019	1	7.42
Mac	2019	2	5.51

Multiple variable combinations in column headers

```
who_df
```

```
# A tibble: 181 x 5
   country             female_pct.obese male_pct.obese female_life.exp male_life.exp
   <chr>                          <dbl>          <dbl>           <dbl>         <dbl>
 1 Afghanistan                      7.6            3.2            64.5          61
 2 Albania                         21.8           21.6            78.6          74.3
 3 Algeria                         34.9           19.9            77.4          75.4
 4 Angola                          12.1            4              64.9          60.3
 5 Antigua and Barbuda             25.9           11.6            77.5          72.5
 6 Argentina                       29             27.3            80.3          73.5
 7 Armenia                         23             17.1            78.1          71.2
 8 Australia                       28.4           29.6            84.8          81
```

Multiple variable combinations in column headers

country	female_pct.obese	male_pct.obese	female_life.exp	male_life.exp
Afghanistan	7.6	3.2	64.5	61
Albania	21.8	21.6	78.6	74.3

country	sex	pct.obese	life.exp
Afghanistan	female	7.6	52.0
Afghanistan	male	3.2	31.0
Albania	female	21.8	7.42
Albania	male	21.6	5.51

The special .value name

```
who_df %>%
  # Example input column name = male_obesity.pct
  pivot_longer(-country,
               names_to = c("sex", ".value"),
               names_sep = "_")
```

```
# A tibble: 362 x 4
   country               sex    pct.obese life.exp
   <chr>                 <chr>      <dbl>    <dbl>
 1 Afghanistan           female       7.6     64.5
 2 Afghanistan           male         3.2     61
 3 Albania               female      21.8     78.6
```

pivot_longer() recap

country	1945	1946
USA	3	2
USSR	NA	NA

country	year	n_bombs
USA	1945	3
USA	1946	2
USSR	1945	NA
USSR	1946	NA

segment	2019_Q1	2019_Q2
iPhone	52.0	31.0
Mac	7.42	5.51

segment	year	quarter	revenue
iPhone	2019	1	52.0
iPhone	2019	2	31.0
Mac	2019	1	7.42
Mac	2019	2	5.51

country	female_pct.obese	male_pct.obese	female_life.exp	male_life.exp
Afghanistan	7.6	3.2	64.5	61
Albania	21.8	21.6	78.6	74.3

country	sex	pct.obese	life.exp
Afghanistan	female	7.6	52.0
Afghanistan	male	3.2	31.0
Albania	female	21.8	7.42
Albania	male	21.6	5.51

Uncounting data

```
nuke_df
```

```
# A tibble: 8 x 2
  country            n_bombs
  <chr>                <int>
1 Pakistan                 2
2 India                    6
3 North Korea              6
4 United Kingdom          21
5 China                   45
6 France                 200
7 Russian Federation   726
8 United States         1150
```

The uncount() function

```
nuke_df %>%
  uncount(n_bombs)
```

```
# A tibble: 2,156 x 1
   country
   <chr>
 1 Pakistan
 2 Pakistan
 3 India
 4 India
 5 India
 6 India
# ... with 2,150 more rows
```

The uncount() function

```
nuke_df %>%
  uncount(2)
```

```
# A tibble: 16 x 2
   country             n_bombs
   <chr>                 <int>
 1 Pakistan                  2
 2 Pakistan                  2
 3 India                     6
 4 India                     6
 5 North Korea               6
 6 North Korea               6
# ... with 10 more rows
```

The uncount() function

```
nuke_df %>%
  uncount(n_bombs, .id = "bomb_id")
```

```
# A tibble: 2,156 x 2
   country     bomb_id
   <chr>         <int>
 1 Pakistan          1
 2 Pakistan          2
 3 India             1
 4 India             2
 5 India             3
 6 India             4
# ... with 2,150 more rows
```

Let's practice!

RESHAPING DATA WITH TIDYR

From long to wide data

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

Variable names in a column

```
who_df
```

```
# A tibble: 362 x 3
   country             metric     value
   <chr>               <chr>      <dbl>
 1 Afghanistan         life_exp    62.7
 2 Afghanistan         pct_obese    5.5
 3 Albania             life_exp    76.4
 4 Albania             pct_obese   21.7
# ... with 358 more rows
```

Variable names in a column

country	metric	value
Afghanistan	life_exp	62.7
Afghanistan	pct_obese	5.5
Albania	life_exp	76.4
Albania	pct_obese	21.7

country	pct_obese	life_exp
Afghanistan	5.5	62.7
Albania	21.7	76.4

The pivot_wider() function

```
who_df %>%
  pivot_wider(names_from = metric, values_from = value)
```

```
# A tibble: 181 x 3
   country               life_exp pct_obese
   <chr>                    <dbl>     <dbl>
 1 Afghanistan               62.7       5.5
 2 Albania                   76.4      21.7
 3 Algeria                   76.4      27.4
 4 Angola                    62.6       8.2
# ... with 177 more rows
```

The pivot_wider() function

```r
who_long_df %>%
  pivot_wider(names_from = metric, values_from = value, names_prefix = "national_")
```

```
# A tibble: 181 x 3
   country                national_life_exp national_pct_obese
   <chr>                              <dbl>              <dbl>
 1 Afghanistan                         62.7                5.5
 2 Albania                             76.4               21.7
 3 Algeria                             76.4               27.4
 4 Angola                              62.6                8.2
# ... with 177 more rows
```

Transposing a data frame

```
sideways_df
```

```
# A tibble: 2 x 5
  variable        `1969` `1970` `1971` `1972`
  <chr>            <int>  <int>  <int>  <int>
1 people_on_moon       4      0      4      4
2 nuclear_bombs       82     85     59     62
```

Transposing a data frame

variable	`1969`	`1970`	`1971`	`1972`
people_on_moon	4	0	4	4
nuclear_bombs	82	85	59	62

year	people_on_moon	nuclear_bombs
1969	4	82
1970	0	85
1971	4	59
1972	4	62

Transposing a data frame: step 1

```
sideways_df %>%
  pivot_longer(-variable, names_to = "year", names_transform = list(year = as.integer))
```

```
# A tibble: 8 x 3
  variable       year  value
  <chr>          <int> <int>
1 people_on_moon 1969      4
2 people_on_moon 1970      0
3 people_on_moon 1971      4
4 people_on_moon 1972      4
5 nuclear_bombs  1969     82
6 nuclear_bombs  1970     85
7 nuclear_bombs  1971     59
8 nuclear_bombs  1972     62
```

Transposing a data frame: step 2

```
sideways_df %>%
  pivot_longer(-variable, names_to = "year", names_transform = list(year = as.integer)) %>%
  pivot_wider(names_from = variable, values_from = value)
```

```
# A tibble: 4 x 3
   year  people_on_moon nuclear_bombs
  <int>           <int>         <int>
1  1969               4            82
2  1970               0            85
3  1971               4            59
4  1972               4            62
```

Let's practice!

RESHAPING DATA WITH TIDYR

Creating unique combinations of vectors

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

The early atomic era: 1945 - 1954

```r
nuke_df
```

```
# A tibble: 13 x 3
   country             year n_bombs
   <chr>              <int>   <int>
 1 United States       1945       3
 2 United States       1946       2
 3 United States       1948       3
 4 Russian Federation  1949       1
 5 Russian Federation  1951       2
 6 United States       1951      16
# ... with 7 more rows
```

The expand_grid() function

```r
full_df <- expand_grid(
  year = 1945:1954,
  country = c(
    "Russian Federation",
    "United Kingdom",
    "United States")
  )

full_df
```

```
# A tibble: 30 x 2
    year country
   <int> <chr>
 1  1945 Russian Federation
 2  1945 United Kingdom
 3  1945 United States
 4  1946 Russian Federation
 5  1946 United Kingdom
 6  1946 United States
 7  1947 Russian Federation
 8  1947 United Kingdom
# ... with 22 more rows
```

right_join() with a tibble of unique combinations

```r
nuke_df %>%
  right_join(
    full_df,
    by = c("country", "year")
  ) %>%
  arrange(year)
```

```
# A tibble: 30 x 3
  country             year n_bombs
  <chr>              <int>   <int>
1 United States       1945       3
2 Russian Federation  1945      NA
3 United Kingdom      1945      NA
4 United States       1946       2
5 Russian Federation  1946      NA
6 United Kingdom      1946      NA
7 Russian Federation  1947      NA
8 United Kingdom      1947      NA
# ... with 22 more rows
```

right_join() with a tibble of unique combinations

```
nuke_df %>%
  right_join(
    full_df,
    by = c("country", "year")
  ) %>%
  arrange(year) %>%
  replace_na(list(n_bombs = 0L))
```

```
# A tibble: 30 x 3
   country             year n_bombs
   <chr>              <int>   <int>
 1 United States       1945       3
 2 Russian Federation  1945       0
 3 United Kingdom      1945       0
 4 United States       1946       2
 5 Russian Federation  1946       0
 6 United Kingdom      1946       0
 7 Russian Federation  1947       0
 8 United Kingdom      1947       0
# ... with 22 more rows
```

anti_join() to select missing observations

```r
full_df %>%
  anti_join(
    nuke_df,
    by = c("country", "year")
  )
```

```
# A tibble: 17 x 2
    year country
   <int> <chr>
 1  1945 Russian Federation
 2  1945 United Kingdom
 3  1946 Russian Federation
 4  1946 United Kingdom
 5  1947 Russian Federation
 6  1947 United Kingdom
 7  1947 United States
 8  1948 Russian Federation
# ... with 9 more rows
```

Let's practice!

RESHAPING DATA WITH TIDYR

Completing data with all value combinations

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

Rolling Stones and Beatles

```
album_df
```

```
# A tibble: 3 x 3
   year artist         n_albums
  <int> <chr>             <int>
1  1977 Beatles               2
2  1977 Rolling Stones        1
3  1979 Beatles               1
```

Initial and target situation

year	artist	n_albums
1977	Beatles	2
1977	Rolling Stones	1
1979	Beatles	1

year	artist	n_albums
1977	Beatles	2
1977	Rolling Stones	1
1979	Beatles	1
1979	Rolling Stones	0

Initial and target situation

year	artist	n_albums
1977	Beatles	2
1977	Rolling Stones	1
1979	Beatles	1

year	artist	n_albums
1977	Beatles	2
1977	Rolling Stones	1
1978	Beatles	0
1978	Rolling Stones	0
1979	Beatles	1
1979	Rolling Stones	0

The complete() function

```
album_df %>%
  complete(year, artist)
```

```
# A tibble: 4 x 3
   year artist         n_albums
  <int> <chr>             <int>
1  1977 Beatles               2
2  1977 Rolling Stones        1
3  1979 Beatles               1
4  1979 Rolling Stones       NA
```

The complete() function: overwriting NA values

```r
album_df %>%
  complete(year, artist, fill = list(n_albums = 0L))
```

```
# A tibble: 4 x 3
   year artist         n_albums
  <int> <chr>             <int>
1  1977 Beatles               2
2  1977 Rolling Stones        1
3  1979 Beatles               1
4  1979 Rolling Stones        0
```

The complete() function: adding unseen values

```
album_df %>%
  complete(
    year,
    artist = c(
      "Beatles",
      "Rolling Stones",
      "ABBA"),
    fill = list(n_albums = 0L)
  )
```

```
# A tibble: 6 x 3
   year artist         n_albums
  <int> <chr>             <int>
1  1977 ABBA                  0
2  1977 Beatles               2
3  1977 Rolling Stones        1
4  1979 ABBA                  0
5  1979 Beatles               1
6  1979 Rolling Stones        0
```

The complete() function: adding unseen values

```r
album_df %>%
  complete(
    year = 1977:1979,
    artist,
    fill = list(n_albums = 0L)
  )
```

```
# A tibble: 6 x 3
   year artist         n_albums
  <int> <chr>             <int>
1  1977 Beatles               2
2  1977 Rolling Stones        1
3  1978 Beatles               0
4  1978 Rolling Stones        0
5  1979 Beatles               1
6  1979 Rolling Stones        0
```

Generating a sequence with full_seq()

```
full_seq(c(1977, 1979), period = 1)
```

```
1977 1978 1979
```

```
full_seq(c(1977, 1979, 1980, 1980, 1980), period = 1)
```

```
1977 1978 1979 1980
```

```
full_seq(album_df$year, period = 1)
```

```
1977 1978 1979
```

Using full_seq() inside complete()

```
album_df %>%
  complete(
    year = full_seq(year, period = 1),
    artist,
    fill = list(n_albums = 0L)
  )
```

```
# A tibble: 6 x 3
   year artist         n_albums
  <dbl> <chr>             <int>
1  1977 Beatles               2
2  1977 Rolling Stones        1
3  1978 Beatles               0
4  1978 Rolling Stones        0
5  1979 Beatles               1
6  1979 Rolling Stones        0
```

Generating a date sequence with full_seq()

```
full_seq(c(as.Date("2000-01-01"), as.Date("2000-01-10")), period = 1)
```

```
 [1] "2000-01-01" "2000-01-02" "2000-01-03" "2000-01-04" "2000-01-05"
 [6] "2000-01-06" "2000-01-07" "2000-01-08" "2000-01-09" "2000-01-10"
```

Let's practice!

RESHAPING DATA WITH TIDYR

Advanced completions

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

Nesting connected variables

```
nuke_df
```

```
# A tibble: 5 x 4
  continent     country n_bombs decade
  <chr>         <chr>     <int>  <int>
1 North America USA           8   1940
2 Europe        USSR          1   1940
3 North America USA         188   1950
4 Europe        USSR         82   1950
5 Europe        UK           21   1950
```

Nesting connected variables

```r
nuke_df %>%
  complete(
    continent,
    country,
    decade,
    fill = list(n_bombs = 0L)
  )
```

```
# A tibble: 12 x 4
   continent     country decade n_bombs
   <chr>         <chr>    <int>   <int>
 1 Europe        UK        1940       0
 2 Europe        UK        1950      21
 3 Europe        USA       1940       0
 4 Europe        USA       1950       0
 5 Europe        USSR      1940       1
 6 Europe        USSR      1950      82
 7 North America UK        1940       0
 8 North America UK        1950       0
# ... with 4 more rows
```

The nesting() function

```
nuke_df %>%
  complete(
    nesting(continent, country),
    decade,
    fill = list(n_bombs = 0L)
  )
```

```
# A tibble: 6 x 4
  continent     country decade n_bombs
  <chr>         <chr>    <int>   <int>
1 Europe        UK        1940       0
2 Europe        UK        1950      21
3 Europe        USSR      1940       1
4 Europe        USSR      1950      82
5 North America USA       1940       8
6 North America USA       1950     188
```

Counting tropical storms

```
storm_df
```

```
# A tibble: 35 x 3
   name       start      end
   <chr>      <date>     <date>
 1 ANDREA     2013-06-05 2013-06-08
 2 ARTHUR     2014-06-28 2014-07-09
 3 ANA        2015-05-06 2015-05-12
 4 BARRY      2013-06-16 2013-06-21
 5 TWO        2014-07-19 2014-07-23
 6 BILL       2015-06-16 2015-06-21
# ... with 29 more rows
```

Counting tropical storms: pivot to long format

```r
storm_df %>%
  pivot_longer(
    -name,
    names_to = "status",
    values_to = "date"
  )
```

```
# A tibble: 70 x 3
   name    status date
   <chr>   <chr>  <date>
 1 ANDREA  start  2013-06-05
 2 ANDREA  end    2013-06-08
 3 ARTHUR  start  2014-06-28
 4 ARTHUR  end    2014-07-09
 5 ANA     start  2015-05-06
 6 ANA     end    2015-05-12
 7 BARRY   start  2013-06-16
 8 BARRY   end    2013-06-21
 9 TWO     start  2014-07-19
10 TWO     end    2014-07-23
# ... with 60 more rows
```

Counting tropical storms: grouped completion

```
storm_df %>%
  pivot_longer(
    -name,
    names_to = "status",
    values_to = "date"
  )  %>%
  group_by(name) %>%
  complete(date = full_seq(date, 1)) %>%
  ungroup()
```

```
# A tibble: 263 x 3
   name   date       status
   <chr>  <date>     <chr>
 1 ANA    2015-05-06 start
 2 ANA    2015-05-07 NA
 3 ANA    2015-05-08 NA
 4 ANA    2015-05-09 NA
 5 ANA    2015-05-10 NA
 6 ANA    2015-05-11 NA
 7 ANA    2015-05-12 end
 8 ANDREA 2013-06-05 start
 9 ANDREA 2013-06-06 NA
10 ANDREA 2013-06-07 NA
# ... with 253 more rows
```

Counting tropical storms: the actual count

```r
storm_df %>%
  pivot_longer(
    -name,
    names_to = "status",
    values_to = "date"
  )  %>%
  group_by(name) %>%
  complete(date = full_seq(date, 1)) %>%
  ungroup() %>%
  count(date, name = "n_storms")
```

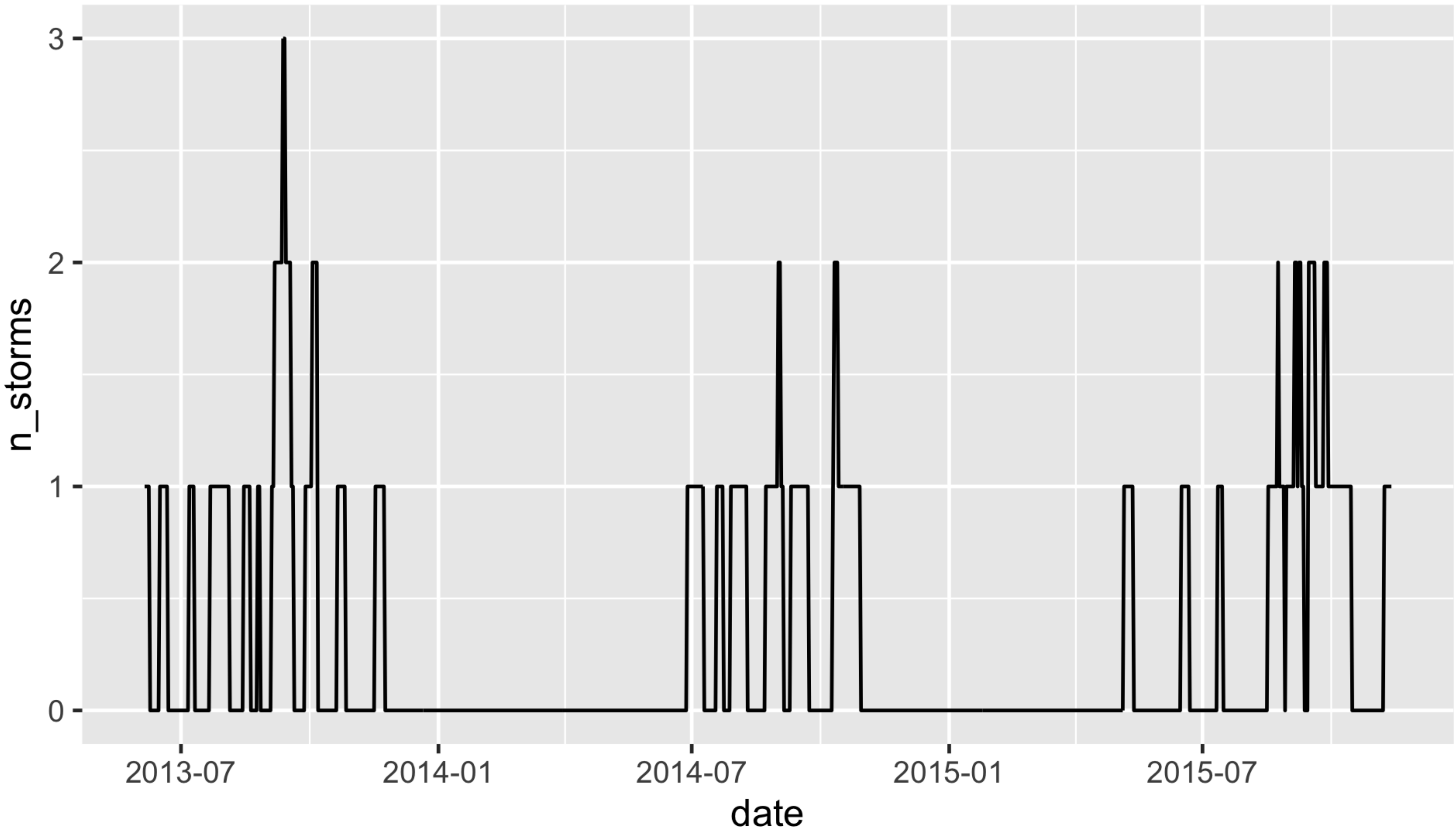
```
# A tibble: 227 x 2
   date       n_storms
   <date>        <int>
 1 2013-06-05        1
 2 2013-06-06        1
 3 2013-06-07        1
 4 2013-06-08        1
 5 2013-06-16        1
 6 2013-06-17        1
 7 2013-06-18        1
 8 2013-06-19        1
 9 2013-06-20        1
10 2013-06-21        1
# ... with 217 more rows
```

Counting tropical storms: adding zero counts

```
storm_df %>%
  pivot_longer(
    -name,
    names_to = "status",
    values_to = "date"
  )  %>%
  group_by(name) %>%
  complete(date = full_seq(date, 1)) %>%
  ungroup() %>%
  count(date, name = "n_storms") %>%
  complete(
    date = full_seq(date, 1),
    fill = list(n_storms = 0L)
  )
```

```
# A tibble: 892 x 2
   date       n_storms
   <date>        <int>
 1 2013-06-05        1
 2 2013-06-06        1
 3 2013-06-07        1
 4 2013-06-08        1
 5 2013-06-09        0
 6 2013-06-10        0
 7 2013-06-11        0
 8 2013-06-12        0
 9 2013-06-13        0
10 2013-06-14        0
# ... with 882 more rows
```

Counting tropical storms: visualizing the result



Timestamp completions

```
sensor_df
```

```
# A tibble: 3 x 2
  time                temperature
  <dttm>                    <int>
1 2020-01-01 11:00:00          25
2 2020-01-01 11:40:00          26
3 2020-01-01 12:20:00          25
```

Timestamp completions

```
sensor_df %>%
  complete(time = seq(from = min(time), to = max(time), by = "20 min"))
```

```
# A tibble: 5 x 2
  time                temperature
  <dttm>                    <int>
1 2020-01-01 11:00:00          25
2 2020-01-01 11:20:00          NA
3 2020-01-01 11:40:00          26
4 2020-01-01 12:00:00          NA
5 2020-01-01 12:20:00          25
```

Timestamp completions

```
sensor_df %>%
  complete(time = seq(from = min(time), to = max(time), by = "20 min")) %>%
  fill(temperature)
```

```
# A tibble: 5 x 2
  time                temperature
  <dttm>                    <int>
1 2020-01-01 11:00:00          25
2 2020-01-01 11:20:00          25
3 2020-01-01 11:40:00          26
4 2020-01-01 12:00:00          26
5 2020-01-01 12:20:00          25
```

Let's practice!

RESHAPING DATA WITH TIDYR

Intro to non-rectangular data

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

Rectangular data

Spreadsheets

	A	B	C
1	name	gender	date
2	Dezik	Male	1951-07-22
3	Dezik	Male	1951-07-29
4	Tsygan	Male	1951-07-22
5	Lisa	Female	1951-07-29
6	Chizhik	Male	1951-08-15

CSV

```
name,gender,date
Dezik,Male,1951-07-22
Dezik,Male,1951-07-29
Tsygan,Male,1951-07-22
Lisa,Female,1951-07-29
Chizhik,Male,1951-08-15
```

Non-rectangular formats

JSON

```json
{
    "name": "Darth Vader",
    "species": "Human",
    "homeworld": "Tatooine",
    "films": [
        "Revenge of the Sith",
        "Return of the Jedi",
        "The Empire Strikes Back",
        "A New Hope"
    ]
}
```

XML

```xml
<note>
  <from>Teacher</from>
  <to>Student</to>
  <heading>Almost there</heading>
  <body>It's the final chapter!</body>
</note>
```

[1] Star Wars data from the repurrrsive package.

A list of lists of lists

```
rjson::fromJSON(file = "star_wars.json")
```

```
[[1]]
[[1]]$name
[1] "Darth Vader"
[[1]]$films
[1] "Revenge of the Sith" "Return of the Jedi" "The Empire Strikes Back" "A New Hope"

[[2]]
[[2]]$name
[1] "Jar Jar Binks"
[[2]]$films
[1] "Attack of the Clones" "The Phantom Menace"
```

A first step to rectangling

```
star_wars_list <- rjson::fromJSON(file = "star_wars.json")
tibble(character = star_wars_list)
```

```
# A tibble: 2 x 1
  character
  <list>
1 <named list [2]>
2 <named list [2]>
```

Unnesting lists to columns

```
tibble(character = star_wars_list) %>%
  unnest_wider(character)
```

```
# A tibble: 2 x 2
  name           films
  <chr>          <list>
1 Darth Vader    <chr [4]>
2 Jar Jar Binks <chr [2]>
```

Unnesting lists to columns

```
tibble(character = star_wars_list) %>%
  unnest_wider(character) %>%
  unnest_wider(films)
```

```
# A tibble: 2 x 5
  name          ...1                 ...2                ...3                    ...4
  <chr>         <chr>                <chr>               <chr>                   <chr>
1 Darth Vader   Revenge of the Sith  Return of the Jedi  The Empire Strikes Back A New Hope
2 Jar Jar Binks Attack of the Clones The Phantom Menace  NA                      NA
```

Let's practice!

RESHAPING DATA WITH TIDYR

From nested values to observations

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

The unnest_wider() function recap

```
tibble(character = star_wars_list) %>%
  unnest_wider(character)
```

```
# A tibble: 2 x 2
  name           films
  <chr>          <list>
1 Darth Vader    <chr [4]>
2 Jar Jar Binks <chr [2]>
```

The unnest_wider() function recap

```
tibble(character = star_wars_list) %>%
  unnest_wider(character) %>%
  unnest_wider(films)
```

```
# A tibble: 2 x 5
  name          ...1                 ...2                ...3                    ...4
  <chr>         <chr>                <chr>               <chr>                   <chr>
1 Darth Vader   Revenge of the Sith  Return of the Jedi The Empire Strikes Back A New Hope
2 Jar Jar Binks Attack of the Clones The Phantom Menace NA                      NA
```

The unnest_longer() function

```
tibble(character = star_wars_list) %>%
  unnest_wider(character) %>%
  unnest_longer(films)
```

```
# A tibble: 45 x 2
   name        films
   <chr>       <chr>
 1 Chewbacca   Revenge of the Sith
 2 Chewbacca   Return of the Jedi
 3 Chewbacca   The Empire Strikes Back
 4 Chewbacca   A New Hope
 5 Chewbacca   The Force Awakens
 6 Darth Vader Revenge of the Sith
 7 Darth Vader Return of the Jedi
 8 Darth Vader The Empire Strikes Back
# ... with 37 more rows
```

Rectangling deeply nested data

```
course_df
```

```
# A tibble: 4 x 2
  ch_id metadata
  <chr> <list>
1 CH1   <named list [3]>
2 CH2   <named list [3]>
3 CH3   <named list [3]>
4 CH4   <named list [3]>
```

Rectangling deeply nested data

```
course_df %>%
  unnest_wider(metadata)
```

```
# A tibble: 4 x 4
  ch_id chapter_title                status      lessons
  <chr> <chr>                        <chr>       <list>
1 CH1   Tidy Data                    Complete    <list [3]>
2 CH2   From Wide to Long and Back Complete    <list [4]>
3 CH3   Expanding Data               Complete    <list [3]>
4 CH4   Rectangling Data             In progress <list [4]>
```

Combining unnest_wider() and unnest_longer()

```
course_df %>%
  unnest_wider(metadata) %>%
  unnest_longer(lessons)
```

```
# A tibble: 14 x 4
   ch_id chapter_title                 status     lessons
   <chr> <chr>                         <chr>      <list>
 1 CH1   Tidy Data                     Complete   <named list [3]>
 2 CH1   Tidy Data                     Complete   <named list [3]>
 3 CH1   Tidy Data                     Complete   <named list [3]>
 4 CH2   From Wide to Long and Back Complete   <named list [3]>
# ... with 10 more rows
```

Digging deeper

```
course_df %>%
  unnest_wider(metadata) %>%
  unnest_longer(lessons) %>%
  unnest_wider(lessons)
```

```
# A tibble: 14 x 6
  ch_id chapter_title                status   l_id  lesson_title                exercises
  <chr> <chr>                        <chr>    <chr> <chr>                       <list>
1 CH1   Tidy Data                    Complete L1    What is tidy data?          <list [2]>
2 CH1   Tidy Data                    Complete L2    Columns with multiple values <list [3]>
3 CH1   Tidy Data                    Complete L3    Missing values              <list [3]>
4 CH2   From Wide to Long and Back Complete L1    From wide to long data      <list [3]>
# ... with 10 more rows
```

And deeper ...

```
course_df %>%
  unnest_wider(metadata) %>%
  unnest_longer(lessons) %>%
  unnest_wider(lessons) %>%
  select(ch_id, l_id, exercises) %>%
  unnest_longer(exercises)
```

```
# A tibble: 41 x 3
  ch_id l_id  exercises
  <chr> <chr> <list>
1 CH1   L1    <named list [2]>
2 CH1   L1    <named list [2]>
3 CH1   L2    <named list [2]>
4 CH1   L2    <named list [2]>
5 CH1   L2    <named list [2]>
6 CH1   L3    <named list [2]>
7 CH1   L3    <named list [2]>
8 CH1   L3    <named list [2]>
# ... with 33 more rows
```

And deeper ...

```
course_df %>%
  unnest_wider(metadata) %>%
  unnest_longer(lessons) %>%
  unnest_wider(lessons) %>%
  select(ch_id, l_id, exercises) %>%
  unnest_longer(exercises) %>%
  unnest_wider(exercises)
```

```
# A tibble: 41 x 4
  ch_id l_id  ex_id complete
  <chr> <chr> <chr> <lgl>
1 CH1   L1    E1    TRUE
2 CH1   L1    E2    TRUE
3 CH1   L2    E1    TRUE
4 CH1   L2    E2    TRUE
5 CH1   L2    E3    TRUE
6 CH1   L3    E1    TRUE
7 CH1   L3    E2    TRUE
8 CH1   L3    E3    TRUE
# ... with 33 more rows
```

Course status update

```
course_df %>%
  unnest_wider(metadata) %>%
  unnest_longer(lessons) %>%
  unnest_wider(lessons) %>%
  select(ch_id, l_id, exercises) %>%
  unnest_longer(exercises) %>%
  unnest_wider(exercises) %>%
  summarize(pct_complete = mean(complete))
```

```
# A tibble: 1 x 1
  pct_complete
         <dbl>
1        0.780
```

Let's practice!

RESHAPING DATA WITH TIDYR

Selecting nested variables

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

Unnesting list columns completely

```
planet_df %>%
  unnest_longer(moons) %>%
  unnest_wider(moons) %>%
  unnest_wider(moon_data)
```

```
# A tibble: 174 x 4
  planet  moon_name radius density
  <chr>   <chr>      <dbl>   <dbl>
1 Mercury NA           NA    NA
2 Venus   NA           NA    NA
3 Earth   Moon       1738.     3.34
4 Jupiter Io         1822.     3.53
5 Jupiter Europa     1561.     3.01
6 Jupiter Ganymede   2631.     1.94
7 Jupiter Callisto   2410.     1.83
8 Jupiter Amalthea     83.4    0.849
# ... with 166 more rows
```

Selective unnesting with hoist()

```
moons :List of 8
 $ :List of 67
  ..$ :List of 2
  .. ..$ moon_name: chr "Io"
  .. ..$ moon_data:List of 2
  .. .. ..$ radius : num 1822
  .. .. ..$ density: num 3.53
```

```
planet_df %>%
  hoist(
    moons,
    first_moon = list(1, "moon_name"),
    radius = list(1, "moon_data", "radius"))
```

```
# A tibble: 8 x 4
  planet  first_moon radius moons
  <chr>   <chr>       <dbl> <list>
1 Mercury NA           NA   <NULL>
2 Venus   NA           NA   <NULL>
3 Earth   Moon       1738.  <list [1]>
4 Jupiter Io         1822.  <list [67]>
5 Mars    Phobos       11.1 <list [2]>
6 Neptune Triton     1353.  <list [14]>
7 Saturn  Mimas       198.  <list [61]>
8 Uranus  Ariel       579.  <list [27]>
```

Selective unnesting with hoist()

```
planet_df %>%
  unnest_longer(moons) %>%
  hoist(
    moons,
    moon_name = "moon_name",
    radius = list("moon_data", "radius")
  )
```

```
# A tibble: 174 x 4
   planet  moon_name radius moons
   <chr>   <chr>      <dbl> <list>
 1 Mercury NA          NA   <NULL>
 2 Venus   NA          NA   <NULL>
 3 Earth   Moon      1738.  <named list [1]>
 4 Jupiter Io        1822.  <named list [1]>
 5 Jupiter Europa    1561.  <named list [1]>
 6 Jupiter Ganymede  2631.  <named list [1]>
 7 Jupiter Callisto  2410.  <named list [1]>
 8 Jupiter Amalthea    83.4 <named list [1]>
 9 Jupiter Himalia     85   <named list [1]>
10 Jupiter Elara       43   <named list [1]>
# ... with 164 more rows
```

Unnesting Google Maps data

```
city_df
```

```
# A tibble: 5 x 2
  city         json
  <chr>        <list>
1 Beijing      <named list [2]>
2 Buenos Aires <named list [2]>
3 New Delhi    <named list [2]>
4 New York     <named list [2]>
5 Paris        <named list [2]>
```

[1] Example from tidyr documentation: https://tidyr.tidyverse.org/articles/rectangle.html

Unnesting Google maps data

```
city_df %>%
  unnest_wider(json)
```

```
# A tibble: 5 x 3
  city         results     status
  <chr>        <list>      <chr>
1 Beijing      <list [1]> OK
2 Buenos Aires <list [1]> OK
3 New Delhi    <list [1]> OK
4 New York     <list [1]> OK
5 Paris        <list [1]> OK
```

[1] Example from tidyr documentation: https://tidyr.tidyverse.org/articles/rectangle.html

Unnesting Google maps data

```
city_df %>%
  unnest_wider(json) %>%
  unnest_longer(results) %>%
  unnest_wider(results)
```

```
  city         address_components formatted_address       geometry
  <chr>        <list>             <chr>                   <list>
1 Beijing      <list [3]>         Beijing, China          <named list [4]>
2 Buenos Aires <list [3]>         Buenos Aires, Argentina <named list [4]>
3 New Delhi    <list [3]>         New Delhi, Delhi, India <named list [4]>
4 New York     <list [3]>         New York, NY, USA       <named list [4]>
5 Paris        <list [4]>         Paris, France           <named list [4]>
# ... with 4 more variables: place_id <chr>, types <list>, partial_match <lgl>, status <chr>
```

[1] Example from tidyr documentation: https://tidyr.tidyverse.org/articles/rectangle.html

Unnesting Google maps data

```
city_df %>%
  unnest_wider(json) %>%
  unnest_longer(results) %>%
  unnest_wider(results) %>%
  unnest_wider(geometry) %>%
  unnest_wider(location) %>%
  select(city, lat, lng)
```

```
# A tibble: 5 x 3
  city           lat    lng
  <chr>        <dbl>  <dbl>
1 Beijing       39.9 116.  
2 Buenos Aires -34.6 -58.4 
3 New Delhi     28.6  77.2 
4 New York      40.7 -74.0 
5 Paris         48.9   2.35
```

[1] Example from tidyr documentation: https://tidyr.tidyverse.org/articles/rectangle.html

Selecting Google maps data with hoist()

```
city_df %>%
  hoist(json,
        lat = list("results", 1, "geometry", "location", "lat"),
        lng = list("results", 1, "geometry", "location", "lng"))
```

```
# A tibble: 5 x 4
  city            lat    lng json
  <chr>         <dbl>  <dbl> <list>
1 Beijing        39.9 116.   <named list [2]>
2 Buenos Aires -34.6 -58.4   <named list [2]>
3 New Delhi      28.6  77.2  <named list [2]>
4 New York       40.7 -74.0  <named list [2]>
5 Paris          48.9   2.35 <named list [2]>
```

[1] Example from tidyr documentation: https://tidyr.tidyverse.org/articles/rectangle.html

Let's practice!

RESHAPING DATA WITH TIDYR

Nesting data for modeling

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

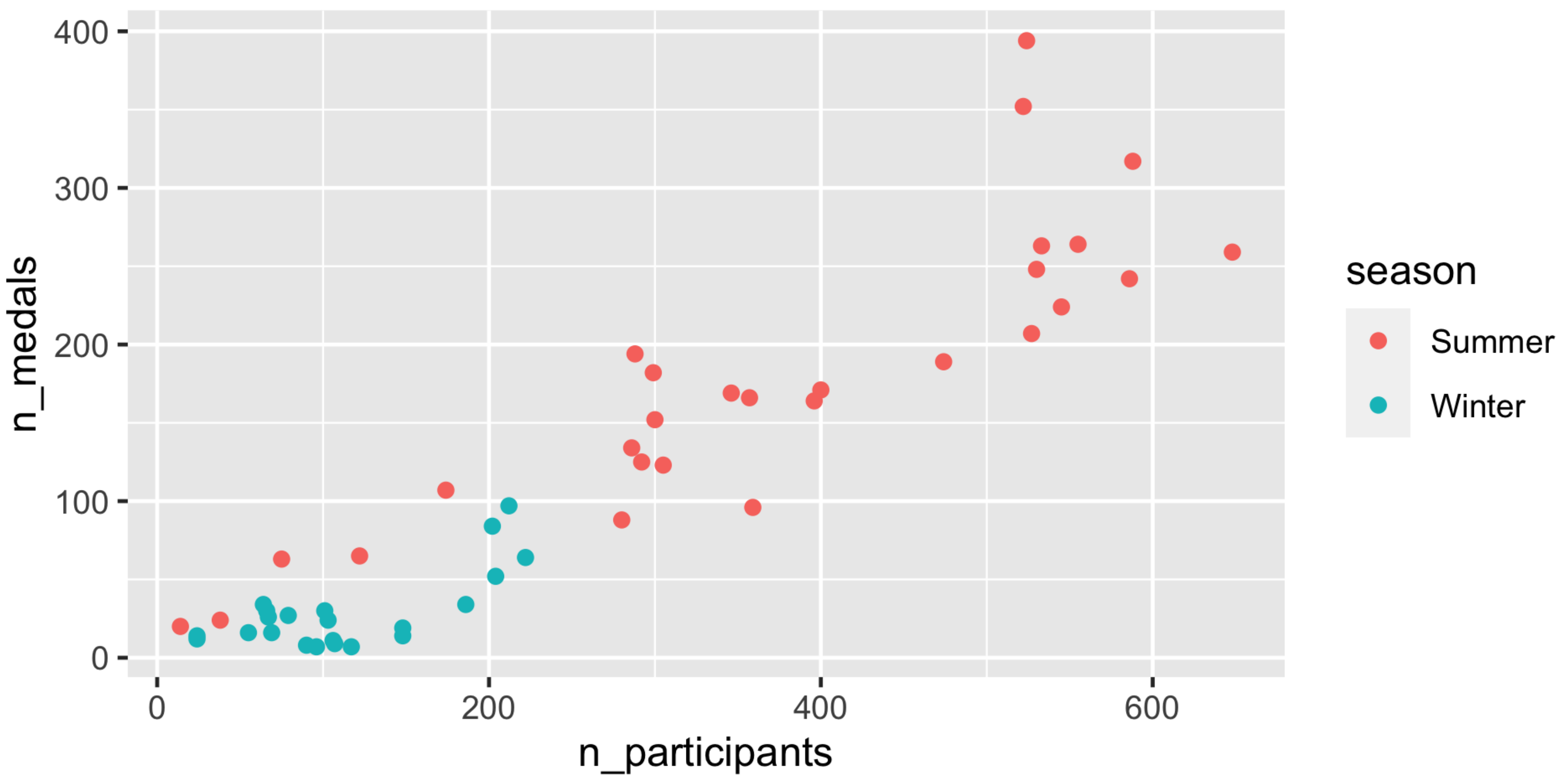
USA Olympic performance

```
usa_olympic_df
```

```
# A tibble: 50 x 5
   country  year season n_participants n_medals
   <chr>   <dbl> <chr>           <int>    <int>
 1 USA      1896 Summer             14       20
 2 USA      1900 Summer             75       63
 3 USA      1904 Summer            524      394
 4 USA      1906 Summer             38       24
 5 USA      1908 Summer            122       65
 6 USA      1912 Summer            174      107
# ... with 44 more rows
```

USA Olympic performance

```r
usa_olympic_df %>%
  ggplot(aes(x = n_participants, y = n_medals, color = season))+
  geom_point()
```



Modeling the pattern

```r
model <- lm(n_medals ~ n_participants + 0, data = usa_olympics_df)

model
```

```
Call:
lm(formula = n_medals ~ n_participants + 0, data = usa_olympics_df)

Coefficients:
n_participants
         0.463
```

Untidy model statistics

```
summary(model)
```

```
Call:
lm(formula = n_medals ~ n_participants + 0, data = usa_olympics_df)
Residuals:
    Min      1Q  Median      3Q     Max
-70.222 -36.175  -9.554   6.871 151.380
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
n_participants  0.46302    0.01791   25.86   <2e-16 ***
--
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 40.17 on 49 degrees of freedom
Multiple R-squared:  0.9317,    Adjusted R-squared:  0.9303
F-statistic: 668.5 on 1 and 49 DF,  p-value: < 2.2e-16
```

The broom package

```
broom::glance(model)
```

```
# A tibble: 1 x 11
  r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC deviance df.residual
      <dbl>         <dbl> <dbl>     <dbl>    <dbl> <int>  <dbl> <dbl> <dbl>    <dbl>       <int>
1     0.932         0.930  40.2      668. 3.25e-30     1  -255.  514.  518.   79079.          49
```

```
broom::tidy(model)
```

```
# A tibble: 1 x 5
  term           estimate std.error statistic  p.value
  <chr>             <dbl>     <dbl>     <dbl>    <dbl>
1 n_participants    0.463    0.0179      25.9 3.25e-30
```

broom + dplyr + tidyr

```
usa_olympics_df %>%
  group_by(country) %>%
  nest()
```

```
# A tibble: 1 x 2
# Groups:   country [1]
  country data
  <chr>   <list>
1 USA     <tibble [50 × 4]>
```

Nested tibble & purrr::map()

```r
usa_olympics_df %>%
  group_by(country) %>%
  nest() %>%
  mutate(fit = purrr::map(data, function(df) lm(n_medals ~ n_participants + 0, data = df)))
```

```
 # A tibble: 1 x 3
# Groups:   country [1]
  country data              fit
  <chr>   <list>            <list>
1 USA     <tibble [50 × 4]> <lm>
```

Working with nested tibbles

```
usa_olympics_df %>%
  group_by(country) %>%
  nest() %>%
  mutate(fit = purrr::map(data, function(df) lm(n_medals ~ n_participants + 0, data = df)),
         glanced = purrr::map(fit, broom::glance))
```

```
 # A tibble: 1 x 4
# Groups:   country [1]
  country data              fit    glanced
  <chr>   <list>            <list> <list>
1 USA     <tibble [50 × 4]> <lm>   <tibble [1 × 11]>
```

Unnesting model results

```r
usa_olympics_df %>%
  group_by(country) %>%
  nest() %>%
  mutate(fit = purrr::map(data, function(df) lm(n_medals ~ n_participants + 0, data = df)),
         glanced = purrr::map(fit, broom::glance)) %>%
  unnest(glanced)
```

```
# A tibble: 1 x 14
# Groups:   country [1]
  country data                fit    r.squared adj.r.squared sigma statistic  p.value    df
  <chr>   <list>              <list>     <dbl>         <dbl> <dbl>     <dbl>    <dbl> <int>
1 USA     <tibble [50 × 4]> <lm>       0.932         0.930  40.2      668. 3.25e-30     1
# with 5 more variables: logLik <dbl>, AIC <dbl>, BIC <dbl> deviance <dbl>, df.residual <int>
```

Unnesting model results

```
usa_olympics_df %>%
  group_by(country) %>%
  nest() %>%
  mutate(fit = purrr::map(data, function(df) lm(n_medals ~ n_participants + 0, data = df)),
         tidied = purrr::map(fit, broom::tidy)) %>%
  unnest(tidied)
```

```
# A tibble: 1 x 8
# Groups:   country [1]
  country data                fit    term           estimate std.error statistic  p.value
  <chr>   <list>              <list> <chr>             <dbl>     <dbl>     <dbl>    <dbl>
1 USA     <tibble [50 × 4]> <lm>   n_participants    0.463    0.0179      25.9 3.25e-30
```

Multiple model pipeline

```
usa_olympics_df %>%
  group_by(country, season) %>%
  nest() %>%
  mutate(fit = purrr::map(data, function(df) lm(n_medals ~ n_participants + 0, data = df)),
         tidied = purrr::map(fit, broom::tidy)) %>%
  unnest(tidied)
```

```
# A tibble: 2 x 9
# Groups:   country, season [2]
  country season data              fit    term           estimate std.error statistic  p.value
  <chr>   <chr>  <list>            <list> <chr>             <dbl>     <dbl>     <dbl>    <dbl>
1 USA     Summer <tibble [28×3]> <lm>   n_participants    0.478    0.0213     22.5  5.29e-19
2 USA     Winter <tibble [22×3]> <lm>   n_participants    0.263    0.0292      9.00 1.18e- 8
```

Let's practice!

RESHAPING DATA WITH TIDYR

Congratulations!

RESHAPING DATA WITH TIDYR

Jeroen Boeye
Head of Machine Learning, Faktion

Separating messy string columns

`separate()`

title	type	duration

title	type	value	unit

`separate_rows()`

drink	ingredients
A	1 2 3
B	1 2

drink	ingredients
A	1
A	2
A	3
B	1
B	2

Pivoting data

`pivot_longer()`

country	1945	1946
USA	3	2
USSR	NA	NA

country	year	n_bombs
USA	1945	3
USA	1946	2
USSR	1945	NA
USSR	1946	NA

`pivot_wider()`

country	metric	value
Afghanistan	life_exp	62.7
Afghanistan	pct_obese	5.5
Albania	life_exp	76.4
Albania	pct_obese	21.7

country	pct_obese	life_exp
Afghanistan	5.5	62.7
Albania	21.7	76.4

Expanding data

`complete()`

year	artist	n_albums
1977	Beatles	2
1977	Rolling Stones	1
1979	Beatles	1

year	artist	n_albums
1977	Beatles	2
1977	Rolling Stones	1
1978	Beatles	0
1978	Rolling Stones	0
1979	Beatles	1
1979	Rolling Stones	0

Unnesting data

```
tibble(character = star_wars_list) %>%
  unnest_wider(character) %>%
  unnest_longer(films)
```

```
# A tibble: 45 x 2
   name        films
   <chr>       <chr>
 1 Chewbacca   Revenge of the Sith
 2 Chewbacca   Return of the Jedi
 3 Chewbacca   The Empire Strikes Back
 4 Chewbacca   A New Hope
 5 Chewbacca   The Force Awakens
 6 Darth Vader Revenge of the Sith
 7 Darth Vader Return of the Jedi
 8 Darth Vader The Empire Strikes Back
# ... with 37 more rows
```

The end

RESHAPING DATA WITH TIDYR