

Wrangling

Basel R Bootcamp
www.therbootcamp.com
[@therbootcamp](https://twitter.com/therbootcamp)

July 2018

Where you're at...

- 1 - Loaded packages (like tidyverse) with `library()`
- 2 - Loaded external files as a new dataframe
- 3 - Explore dataframes
- 4 - Calculate descriptive statistics on specific columns

What's next?... **Wrangling!**

```
# Step 0) Load libraries

library(tidyverse)

# Step 1) Read file called baslers.txt
# in a data folder with read_csv()
# and save as new object baslers

baslers <- read_csv(file = "data/baslers.txt")

# Step 2) Explore data

View(baslers)    # Open in new window
dim(baslers)     # Show number of rows and columns
names(baslers)   # Show names

# Step 3) Calculate descriptives on named columns

mean(baslers$age) # What is the mean age?
table(baslers$sex) # How many of each sex?

# Step 4) ...
```

What is wrangling?

Transform

Change column names

Add new columns

Organise

Sort data by columns

Merging data from two separate dataframes

Move data between columns and rows

Aggregate

Group data and summarise

Transform

id	time1	time2
1	62	60
2	59	45
3	64	50

→
“Add Change column”
“Convert time1 to minutes”

id	time1	time2	change	time1_min
1	62	60	-2	1.03
2	59	45	-6	0.98
3	64	50	-14	1.06

Organise

id	time1	time2
1	62	60
2	59	45
3	64	50

→
“Convert rows to columns”
“Order rows by id and time”

id	time	x
1	1	62
2	1	59
3	1	64
1	2	60
2	2	45
3	2	50

Aggregate

id	time	x
1	1	62
2	1	59
3	1	64
1	2	60
2	2	45
3	2	50

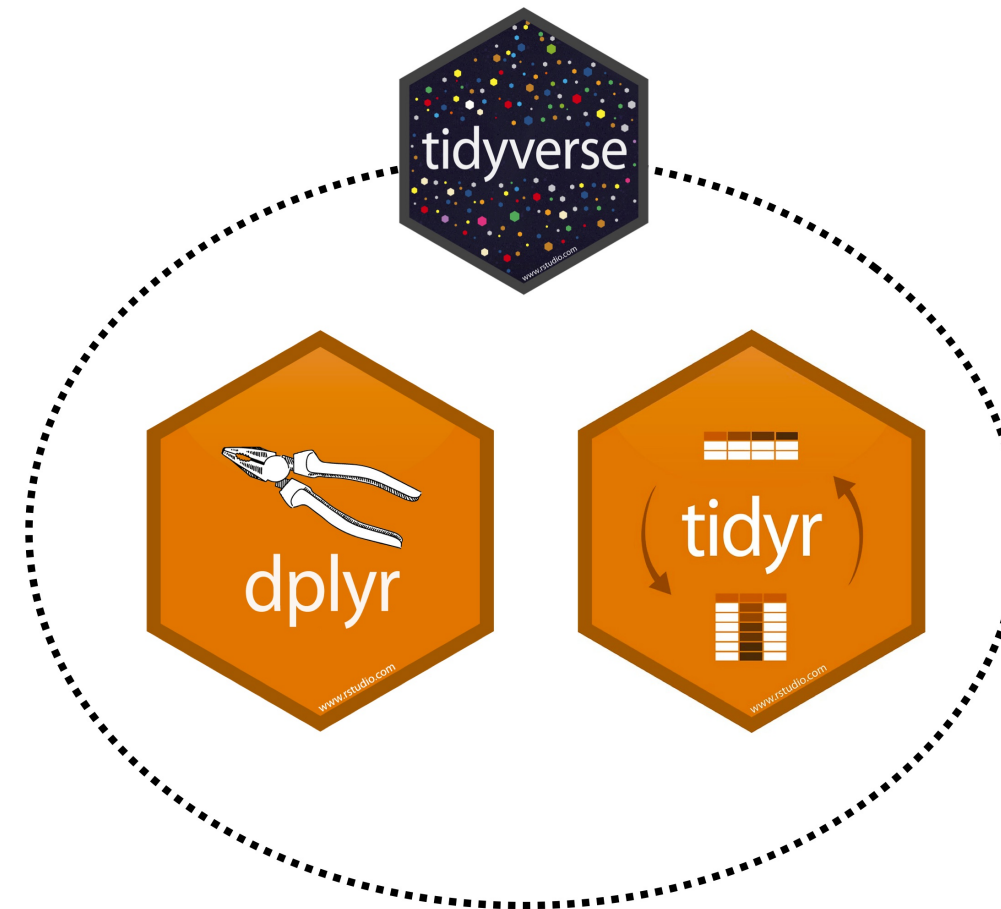
→
“Group by Time”
“Calculate mean and standard deviation”

time	mean	sd
1	61.66	60
2	51.66	45

dplyr & tidyr

To wrangle data in R, we will use the `dplyr` and `tidyr` packages.

```
# Load packages individually  
  
# install.packages('dplyr')  
# install.packages('tidyr')  
  
library(dplyr)  
library(tidyr)  
  
# Or just use the tidyverse!  
  
# install.packages('tidyverse')  
  
library(tidyverse)
```



The Pipe! %>%

dplyr makes extensive use of a new operator called the "Pipe" %>%

Read the "Pipe" %>% as "And Then..."

```
# Start with data  
data %>% # AND THEN...  
  
DO_SOMETHING %>% # AND THEN...  
DO_SOMETHING %>% # AND THEN...  
DO_SOMETHING %>% # AND THEN...
```



This is not a pipe (but %>% is!)

The Pipe! %>%

Task: Calculate the mean of a vector of scores

```
# Create a vector score  
score <- c(8, 4, 6, 3, 7, 3)
```

Base-R method

```
mean(x = score)
```

```
## [1] 5.167
```

Pipe %>% method

```
score %>% # AND THEN  
  mean()
```

```
## [1] 5.167
```

FUN(OBJECT, ...)

Is the same thing as...

OBJECT %>% FUN(__ , ...)



The **OBJECT** to the left of the pipe
%>% becomes the first argument to
the **FUN()** to the right of the pipe

The Pipe! %>%

Task: Calculate the mean of a vector of scores and round to 1 digit.

```
# Create a vector score  
score <- c(8, 4, 6, 3, 7, 3)
```

Base-R method

```
round(x = mean(score), digits = 1)
```

```
## [1] 5.2
```

Pipe %>% method

```
score %>%      # AND THEN  
  mean() %>%   # AND THEN  
  round(digits = 1)
```

```
## [1] 5.2
```

FUN(OBJECT, ...)

Is the same thing as...

OBJECT %>% FUN(__ , ...)



The **OBJECT** to the left of the pipe
%>% becomes the first argument to
the **FUN()** to the right of the pipe

dplyr Functions

There are **dozens of wrangling functions** in dplyr.

For an overview, check out dplyr.tidyverse.org

Summarise Data	Make New Variables	Combine Data Sets
<p>Summary Functions</p> <p><code>dplyr::summarise(iris, avg = mean(Sepal.Length))</code> Summarise data into single row of values.</p> <p><code>dplyr::summarise_each(iris, fun = mean())</code> Apply summary function to each column.</p> <p><code>dplyr::count(iris, Species, wt = Sepal.Length)</code> Count number of rows with each unique value of variable (with or without weights).</p> <p>Summary Functions</p> <p><code>dplyr::first</code> First value of a vector.</p> <p><code>dplyr::last</code> Last value of a vector.</p> <p><code>dplyr::nth</code> nth value of a vector.</p> <p><code>dplyr::n</code> # of values in a vector.</p> <p><code>dplyr::n_distinct</code> # of distinct values in a vector.</p> <p><code>dplyr::IQR</code> IQR of a vector.</p> <p>min Minimum value in a vector.</p> <p>max Maximum value in a vector.</p> <p>mean Mean value of a vector.</p> <p>median Median value of a vector.</p> <p>var Variance of a vector.</p> <p>sd Standard deviation of a vector.</p>	<p>Mutate</p> <p><code>dplyr::mutate(iris, sepal = Sepal.Length + Sepal.Width)</code> Compute and append one or more new columns.</p> <p><code>dplyr::mutate_each(iris, fun = min_rank())</code> Apply window function to each column.</p> <p><code>dplyr::transmute(iris, sepal = Sepal.Length + Sepal.Width)</code> Compute one or more new columns. Drop original columns.</p> <p>Window Functions</p> <p><code>dplyr::lead</code> Copy with values shifted by 1.</p> <p><code>dplyr::lag</code> Copy with values lagged by 1.</p> <p><code>dplyr::dense_rank</code> Ranks with no gaps.</p> <p><code>dplyr::min_rank</code> Ranks. Ties get min rank.</p> <p><code>dplyr::percent_rank</code> Ranks rescaled to [0, 1].</p> <p><code>dplyr::row_number</code> Ranks. Ties get to first value.</p> <p><code>dplyr::ntile</code> Bin vector into buckets.</p> <p><code>dplyr::between</code> Are values between a and b?</p> <p><code>dplyr::cume_dist</code> Cumulative distribution.</p> <p>Cumulative Functions</p> <p><code>dplyr::cumall</code> Cumulative all.</p> <p><code>dplyr::cumany</code> Cumulative any.</p> <p><code>dplyr::cummean</code> Cumulative mean.</p> <p><code>dplyr::cumsum</code> Cumulative sum.</p> <p><code>dplyr::cummax</code> Cumulative max.</p> <p><code>dplyr::cummin</code> Cumulative min.</p> <p><code>dplyr::cumprod</code> Cumulative prod.</p> <p><code>dplyr::pmax</code> Element-wise max.</p> <p><code>dplyr::pmin</code> Element-wise min.</p>	<p>Joining</p> <p><code>dplyr::left_join(a, b, by = "x1")</code> Join matching rows from b to a.</p> <p><code>dplyr::right_join(a, b, by = "x1")</code> Join matching rows from a to b.</p> <p><code>dplyr::inner_join(a, b, by = "x1")</code> Join data. Retain only rows in both sets.</p> <p><code>dplyr::full_join(a, b, by = "x1")</code> Join data. Retain all values, all rows.</p> <p>Semantic Joins</p> <p><code>dplyr::semi_join(a, b, by = "x1")</code> All rows in which b has a match in a.</p> <p><code>dplyr::anti_join(a, b, by = "x1")</code> All rows in a that do not have a match in b.</p> <p>Set Operations</p> <p><code>dplyr::intersect(x, y)</code> Rows that appear in both x and y.</p> <p><code>dplyr::union(x, y)</code> Rows that appear in either or both x and y.</p> <p><code>dplyr::setdiff(x, y)</code> Rows that appear in x but not y.</p> <p>Other Functions</p> <p><code>dplyr::bind_rows(x, y)</code> Append x to y as new rows.</p> <p><code>dplyr::bind_cols(x, y)</code> Append x to y as new columns. Caution: matches rows by position.</p>

Wrangling Cheat Sheet

1) Change column name to sex

```
df %>%
  rename(sex = b)
```

2) Add new bmi column

```
df %>%
  mutate(bmi = weight / height ^ 2))
```

id	sex b	weight	height	bmi
1	F	46	1.65	27.88
2	F	36	1.76	20.45
3	F	44	1.57	28.03
4	M	38	1.78	21.35
5	M	35	1.87	18.72

3) Only females

```
df %>%
  filter(sex == "F")
```

```
df %>%
  rename(sex = b) %>%
  mutate(bmi = weight / height ^ 2)) %>%
  filter(sex == "F")
```

All 3 Steps at once!

Transformation Functions

Function	Description
<code>rename()</code>	Change column names
<code>mutate()</code>	Create a new column from existing columns
<code>case_when()</code>	Recode values from a vector to another
<code>left_join()</code>	Combine multiple dataframes

```
patients_df    # Demographic data
```

```
## # A tibble: 5 x 3
##   id      b      c
##   <dbl> <dbl> <dbl>
## 1     1    37     1
## 2     2    65     2
## 3     3    57     2
## 4     4    34     1
## 5     5    45     2
```

rename

Change **column names** with `rename()`.

```
df %>%  
  rename(NEW = OLD,  
         NEW = OLD)  
  
patients_df  # Original
```

```
## # A tibble: 5 x 3  
##       id      b      c  
##   <dbl> <dbl> <dbl>  
## 1     1    37     1  
## 2     2    65     2  
## 3     3    57     2  
## 4     4    34     1  
## 5     5    45     2
```

Change the old name "b" to "age", and "c" to "arm"

```
# 0) Start with patients_df data  
patients_df %>%  
  
# 1) Change column names with rename()  
  rename(age = b,  # New = Old  
         arm = c)  # New = Old
```

```
## # A tibble: 5 x 3  
##       id  age  arm  
##   <dbl> <dbl> <dbl>  
## 1     1    37     1  
## 2     2    65     2  
## 3     3    57     2  
## 4     4    34     1  
## 5     5    45     2
```

mutate

Calculate **new columns**, or change existing ones, with `mutate()`.

```
df %>%
  mutate(
    NEW1 = DEFINITION1,
    NEW2 = DEFINITION2,
    NEW3 = DEFINITION3,
    ...
  )
```

Calculate two new columns `age_months` and `age_years`

```
patients_df %>%
  rename(age = b,
         arm = c) %>%   # AND THEN...

# Create new columns with mutate()
mutate(age_months = age * 12,
       age_decades = age / 10)
```

```
## # A tibble: 5 x 5
##   id    age arm age_months age_decades
##   <dbl> <dbl> <dbl>      <dbl>      <dbl>
## 1     1    37     1        444         3.7
## 2     2    65     2        780         6.5
## 3     3    57     2        684         5.7
## 4     4    34     1        408         3.4
## 5     5    45     2        540         4.5
```

case_when

Use `case_when()` with `mutate()` to define **new columns based on logical conditions**.

```
# Using mutate(case_when())
df %>%
  mutate(
    NEW = case_when(
      COND1 ~ VAL1,
      COND2 ~ VAL2
    )
  )
```

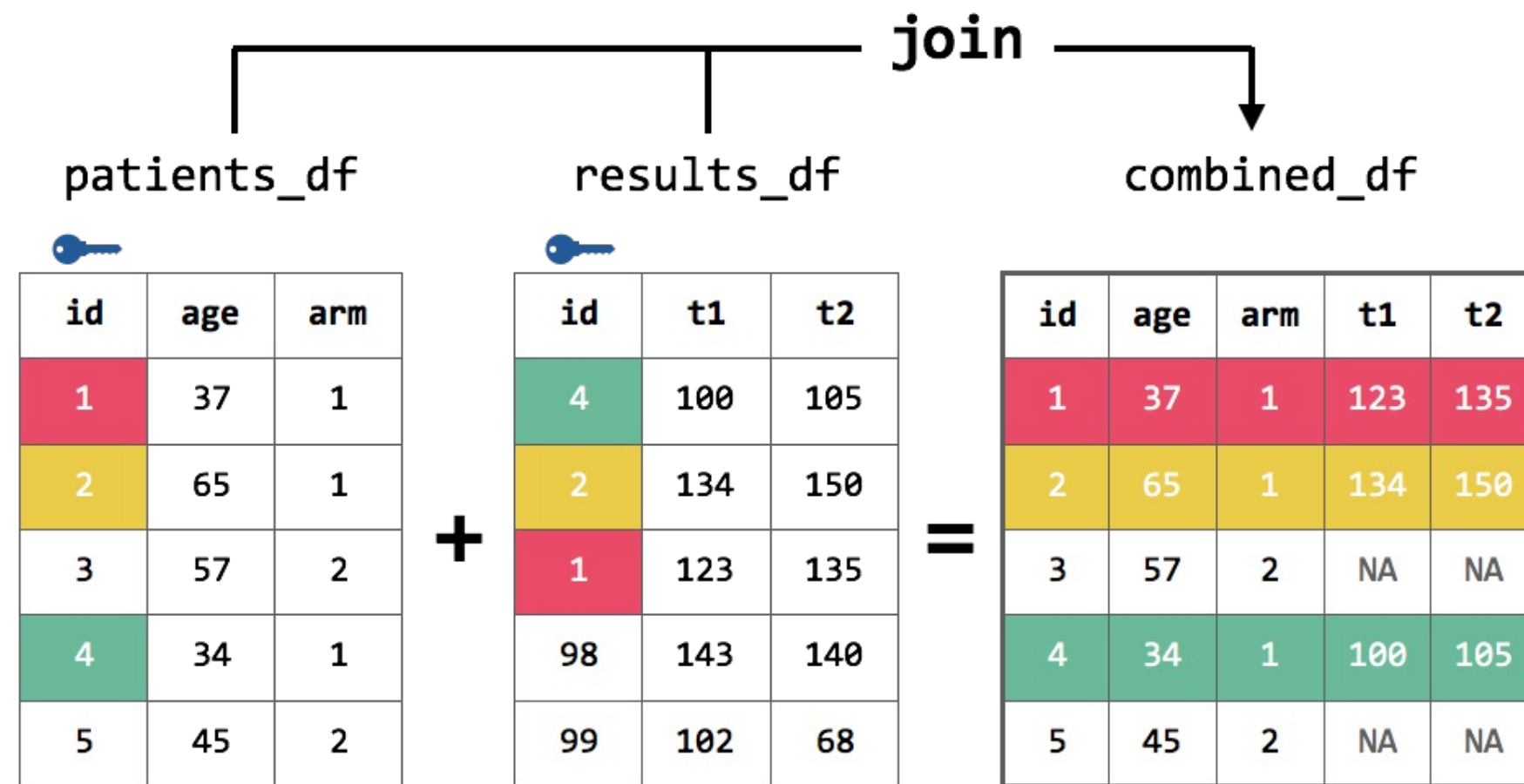
```
patients_df %>%
  rename(age = b,
         arm = c) %>%

  # Create column arm_char based on values of arm

  mutate(arm_char = case_when(arm == 1 ~ "placebo",
                              arm == 2 ~ "drug"))
```

```
## # A tibble: 5 x 4
##   id    age    arm arm_char
##   <dbl> <dbl> <dbl> <chr>
## 1     1    37     1 placebo
## 2     2    65     2 drug
## 3     3    57     2 drug
## 4     4    34     1 placebo
## 5     5    45     2 drug
```

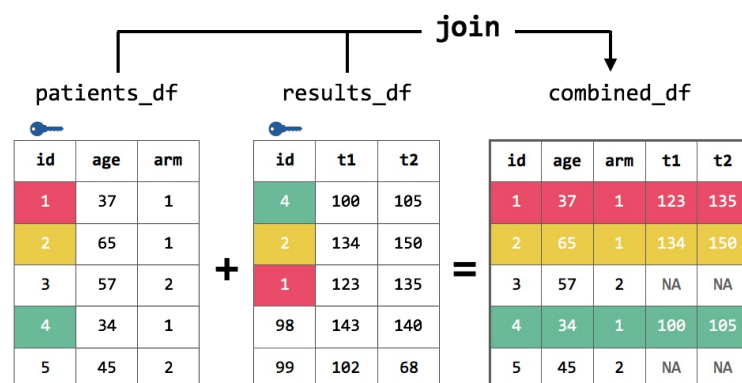
Joining data



left_join

Use `left_join()` to **combine two data frames** based on one or more key columns

```
# Join df2 to df1
# using KEY as the key column
df1 %>%
  left_join(df2,
            by = c("KEY"))
```



```
# Join patients_df with results_df to create combined_df
combined_df <- patients_df %>%
  rename(age = b, arm = c) %>%
  mutate(arm_char = case_when(arm == 1 ~ "placebo",
                              arm == 2 ~ "drug")) %>%
```

```
# Join with results_df with left_join()
left_join(results_df, by = "id")
```

```
# Show a few columns
combined_df %>%
  select(id, arm, age, t1, t2)
```

```
## # A tibble: 5 x 5
##   id    arm    age    t1    t2
##   <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1     1     37    123    135
## 2     2     2     65    143    140
## 3     3     2     57     NA     NA
## 4     4     1     34    100    105
## 5     5     2     45     NA     NA
```

Keep in mind

- 1 - Don't forget to start by assigning to a new (or existing) object with <-
- 2 - Keep adding new functions connected by the pipe %>%
- 3 - Order matters! You can refer to new columns in later code

```
# 0) Start with patients_df data
combined_df <- patients_df %>%

# 1) Change column names with rename()
  rename(age = b,
         arm = c) %>% # AND THEN...

# 2) Create new columns with mutate()
  mutate(age_months = age * 12,
         age_decades = age / 10,
         arm_char = case_when(arm == 0 ~ "placebo",
                              arm == 1 ~ "drug")
  ) %>% # AND THEN..

# 3) Add data from results_df with left_join()
  left_join(results_df, by = "id")
```

Organisation Functions

Organisation functions help you shuffle your data by **sorting rows** by columns, **filter rows** based on criteria, **select columns** (etc).

Function	Purpose	Example
arrange()	Sort rows by columns	df %>% arrange(arm, age)
slice()	Select rows by location	df %>% slice(1:10)
filter()	Select specific rows by criteria	df %>% filter(age > 50)
select()	Select specific columns	df %>% select(arm, t1)

arrange

Use `arrange()` to **arrange (aka, sort) rows** in increasing or decreasing order of one (or more) columns.

To sort in descending order, use `desc()`

```
# Sort combined_df by id  
# then in descending order of age  
combined_df %>%  
  arrange(id, desc(age))
```

```
## # A tibble: 5 x 6  
##       id    age  arm arm_char    t1    t2  
##   <dbl> <dbl> <dbl> <chr>    <dbl> <dbl>  
## 1     1    37     1 placebo   123   135  
## 2     2    65     2 drug     143   140  
## 3     3    57     2 drug      NA    NA  
## 4     4    34     1 placebo   100   105  
## 5     5    45     2 drug      NA    NA
```

filter

Use `filter()` to **select rows** (and remove others) based on criteria

For complex conditions, chain multiple logical comparison operators with `&` (AND) and `|` (OR)

`==` - is equal to

`<`, `>` - smaller/greater than

`≤`, `≥` - smaller/greater than or equal

`&`, `&&` - logical AND

`|`, `||` - logical OR

Select patients over 30 given drug.

```
# Filter patients older than 30 given drug
combined_df %>%
  arrange(id, desc(age)) %>%
  filter(age > 30 & arm_char == "drug")
```

```
## # A tibble: 3 x 6
##   id    age    arm arm_char    t1    t2
##   <dbl> <dbl> <dbl> <chr>    <dbl> <dbl>
## 1     2    65     2 drug      143   140
## 2     3    57     2 drug      NA    NA
## 3     5    45     2 drug      NA    NA
```

select

Use `select()` to **select columns** (and remove all others)

```
# Select columns id, age
df %>%
  select(id, age)
```

Remove columns with `-`.

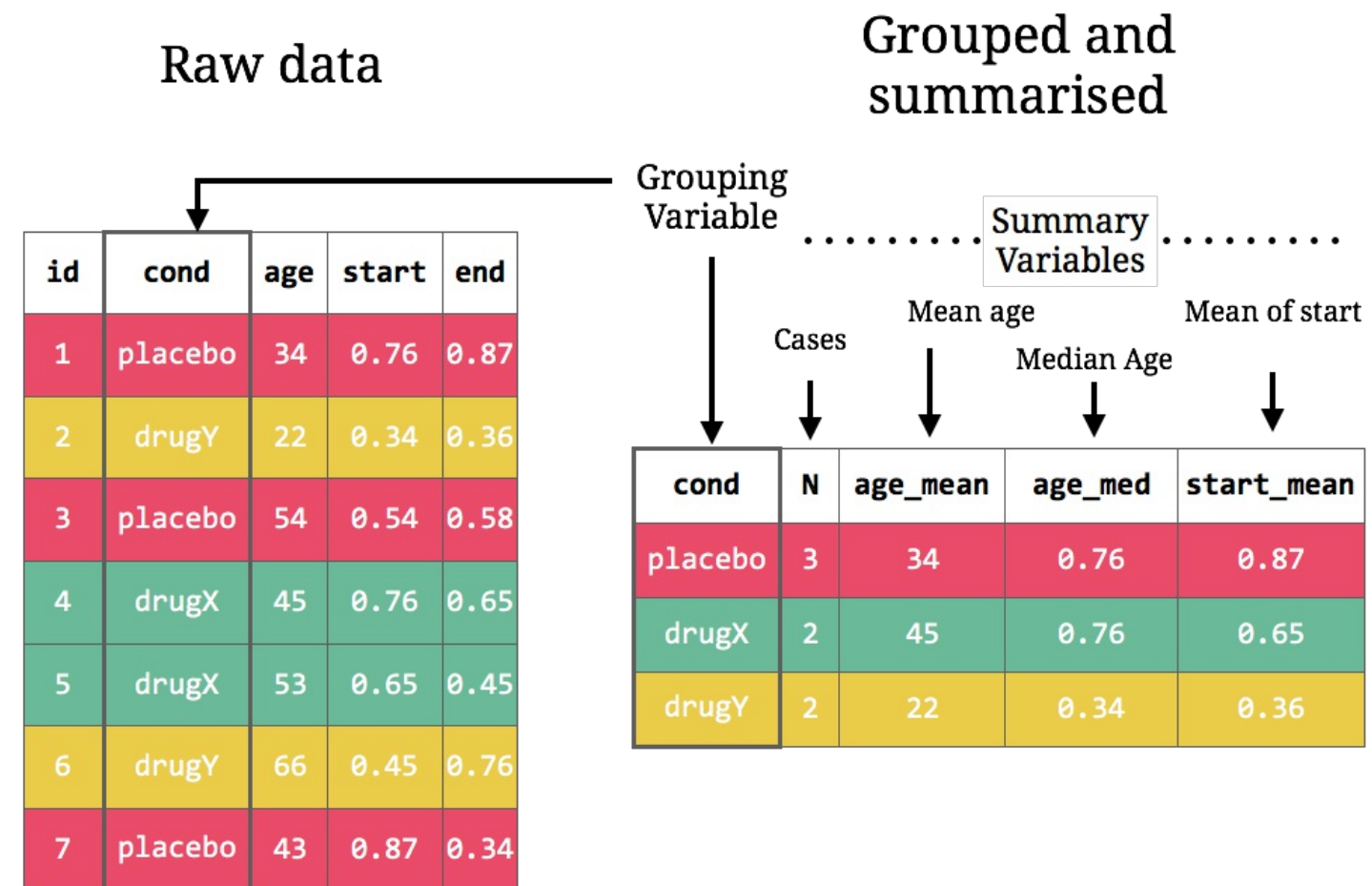
```
# Select everything BUT sex and id
df %>%
  select(-sex, -id)
```

Remove `age_months` and `age_decades` columns.

```
combined_df %>%
  arrange(id, desc(age)) %>%
  filter(arm_char == "drug" & age > 30) %>%
  # Drop age and arm columns
  select(-age, -arm)
```

```
## # A tibble: 3 x 4
##   id arm_char    t1    t2
##   <dbl> <chr>    <dbl> <dbl>
## 1     2 drug      143   140
## 2     3 drug       NA    NA
## 3     5 drug       NA    NA
```

Grouped Aggregation



group_by, summarise

Use `group_by()` to **group data** according to one or more columns

After grouping data, use `summarise()` to **calculate summary statistics** across groups of data

Statistical functions

Function	asdf
<code>n()</code>	Number of cases in each group
<code>mean()</code> , <code>median()</code> , <code>max()</code> , <code>min()</code> <code>sum()</code>	Summary stats

```
# Group data by arm, and calculate many
# summary statistics
combined_df %>%
  group_by(arm) %>%
  summarise(
    N = n(),
    age_mean = mean(age),
    t1_mean = mean(t1, na.rm = TRUE),
    t2_mean = mean(t2, na.rm = TRUE)
  )
```

```
## # A tibble: 2 x 5
##   arm      N age_mean t1_mean t2_mean
##   <chr> <int>   <dbl>   <dbl>   <dbl>
## 1 drug      3    55.7    143    140
## 2 placebo   2    35.5    112    120
```

Reshaping data

Two key functions that allow you to **reshape** a dataframe between 'wide' and 'long' formats.

Some functions require data to be in a certain shape.

Two key tidyr functions

Function	Result
gather()	Move data from 'wide' to 'long' format
spread()	Move data from 'long' to 'wide' format

'Wide' vs. 'Long' data

```
# Wide format
stock_w
```

```
##   id t1 t2
## 1  a 10 20
## 2  b 20 26
## 3  c 15 30
```

```
# Long format
stock_l
```

```
##   id time measure
## 1  a   t1      10
## 2  b   t1      20
## 3  c   t1      15
## 4  a   t2      20
## 5  b   t2      26
## 6  c   t2      30
```

gather

```
# Show wide data  
stock_w
```

```
##   id t1 t2  
## 1  a 10 20  
## 2  b 20 26  
## 3  c 15 30
```

```
# "Gather" wide data to long  
stock_w %>%  
  gather(time,      # New group column  
          measure,  # New target column  
          -id)      # ID column
```

```
##   id time measure  
## 1  a  t1      10  
## 2  b  t1      20  
## 3  c  t1      15  
## 4  a  t2      20  
## 5  b  t2      26  
## 6  c  t2      30
```

spread

```
# Show long data  
stock_l
```

```
##   id time measure  
## 1  a  t1      10  
## 2  b  t1      20  
## 3  c  t1      15  
## 4  a  t2      20  
## 5  b  t2      26  
## 6  c  t2      30
```

```
# "Spread" long data to wide  
stock_l %>%  
  spread(time,      # Old group column  
          measure) # Old target column
```

```
##   id t1 t2  
## 1  a 10 20  
## 2  b 20 26  
## 3  c 15 30
```

Summary

- 1 - Start by assigning your result to a new object to save it!
- 2 - "Keep the pipe `%>%` going" to continue working with your data frame.
- 3 - The output of dplyr functions will (almost) always be a **tibble**.
- 4 - You can almost always include **multiple operations** within each function.

```
# Assign result to baslers_agg
baslers_agg <- baslers %>%

# Change column names with rename()
rename(age_years = age,
        weight_kg = weight) %>% # PIPE!

# Select specific rows with filter()
filter(age_years < 40) %>% # PIPE!

# Create new columns with mutate()
mutate(debt_ratio = debt / income) %>% # PIPE!

# Group data with group_by()
group_by(sex) %>% # PIPE!

# Calculate summary statistics with summarise()
summarise(income_mean = mean(income),
          debt_mean = mean(debt),
          dr_mean = mean(dr))
```


Practical

[Link to practical](#)