

Wrangling

Introduction to Data Science with R

www.therbootcamp.com

@therbootcamp

October 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 and summarise

Group data and calculate and summary stats

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 |

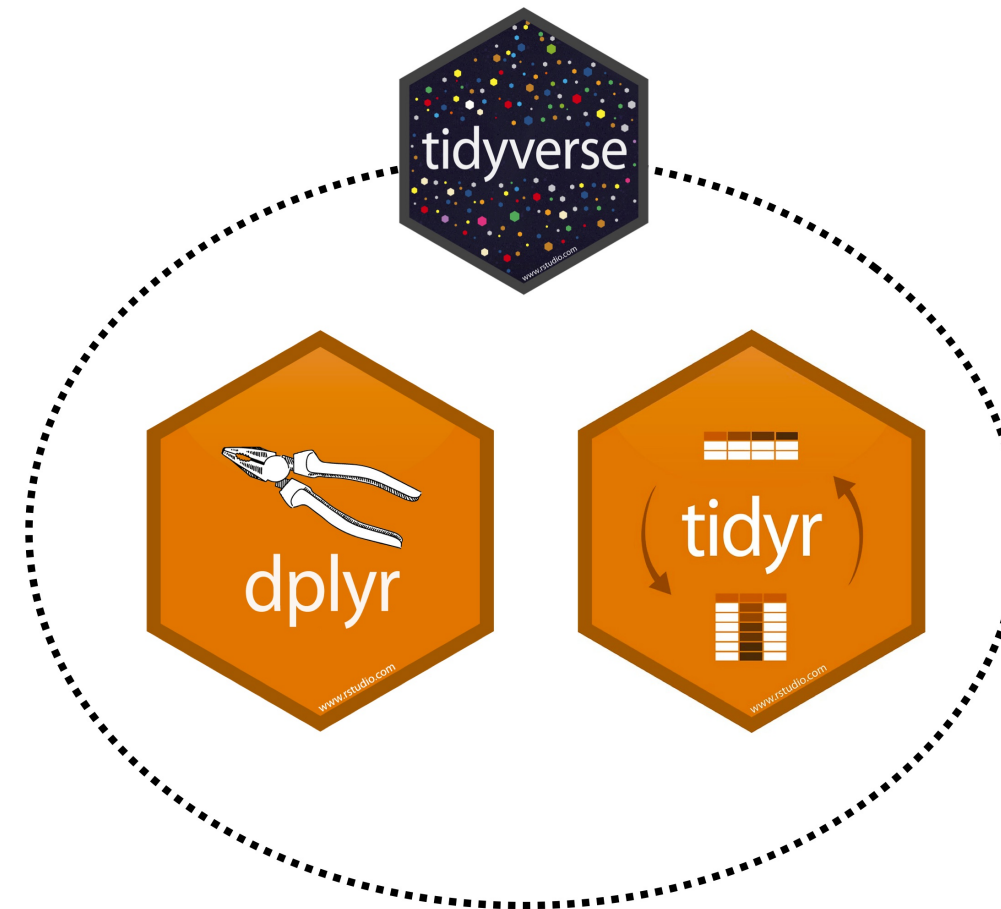
→
“Group by Time”
“Calculate mean and standard deviation”

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

All 3 Steps at once!

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


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_decades`

```
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
    )
  )
```

Create `arm_char`, which shows arm as a meaningful character rather than an integer.

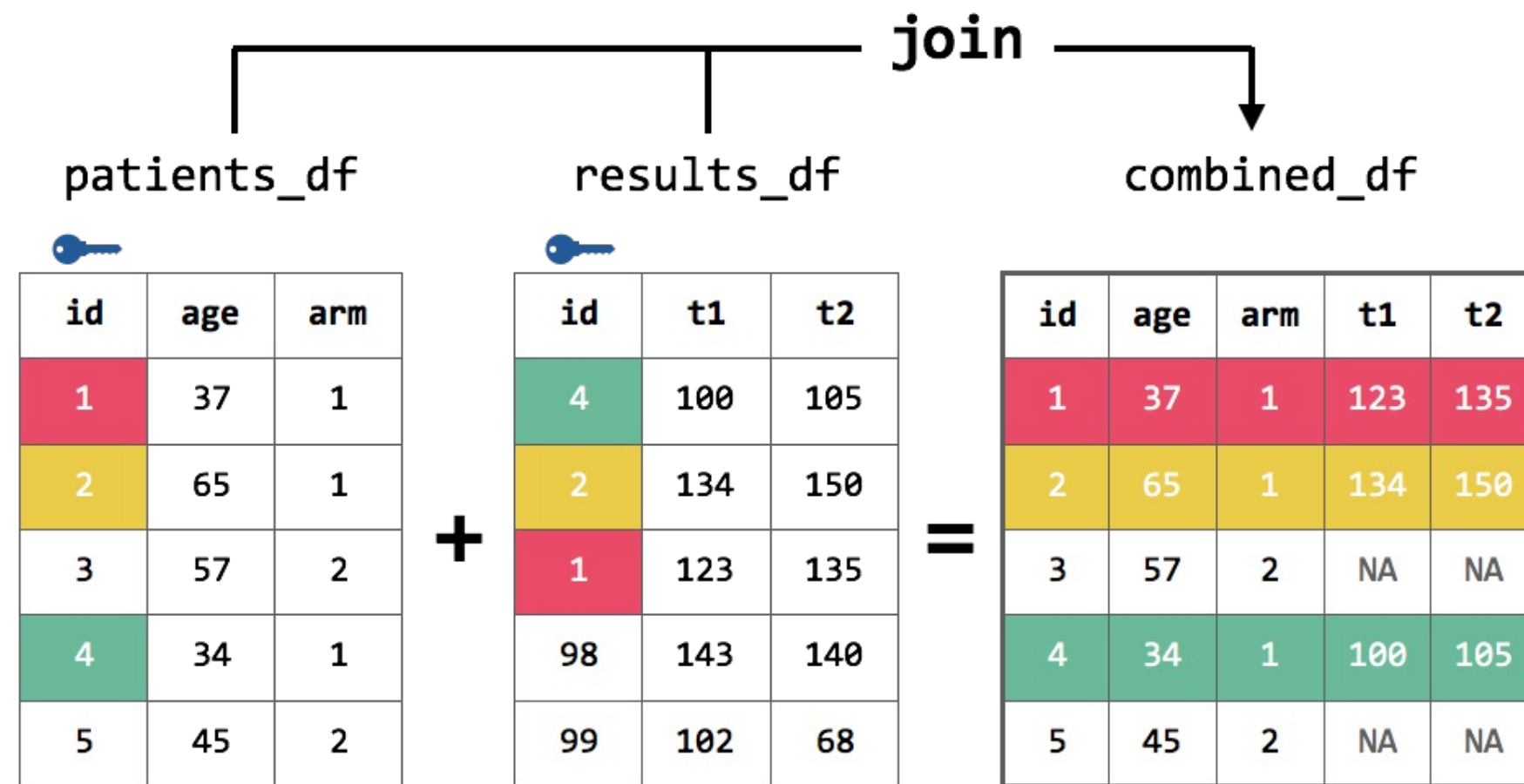
```
patients_df %>%
  rename(age = b,
         arm = c) %>%

  # 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
```

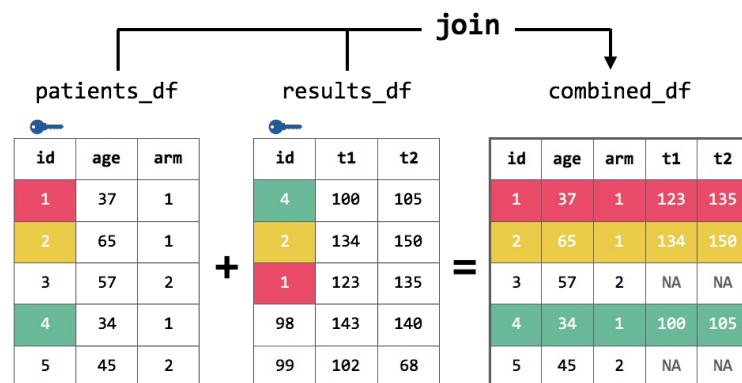
Joining data



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")
```

Quiz 1

Do these two chunks do the same thing?

Chunk A

```
baselers <- baselers %>%  
  rename(salary = income)  
  
baselers <- baselers %>%  
  rename(weight = weight_kg)
```

Chunk B

```
baselers <- baselers %>%  
  rename(salary = income,  
         weight = weight_kg)
```


Quiz 1

Do these two chunks do the same thing?

Chunk A

```
baselers <- baselers %>%  
  rename(salary = income)  
  
baselers <- baselers %>%  
  rename(weight = weight_kg)
```

Answer: Yes!

Chunk B

```
baselers <- baselers %>%  
  rename(salary = income,  
         weight = weight_kg)
```

Quiz 2

Do these two chunks do the same thing?

Chunk A

```
baselers <- baselers %>%  
  rename(salary = income)  
  
baselers <- baselers %>%  
  mutate(age_months = age * 12)
```

Chunk B

```
baselers <- baselers %>%  
  rename(salary = income) %>%  
  mutate(age_months = age * 12)
```

Quiz 2

Do these two chunks do the same thing?

Chunk A

```
baselers <- baselers %>%  
  rename(salary = income)  
  
baselers <- baselers %>%  
  mutate(age_months = age * 12)
```

Answer: Yes!

Chunk B

```
baselers <- baselers %>%  
  rename(salary = income) %>%  
  mutate(age_months = age * 12)
```

Quiz 3

Do these two chunks do the same thing?

Chunk A

```
baselers %>%  
  rename(salary = income) %>%  
  mutate(age_months = age * 12)
```

Chunk B

```
baselers <- baselers %>%  
  rename(salary = income) %>%  
  mutate(age_months = age * 12)
```

Quiz 3

Do these two chunks do the same thing?

Chunk A

```
baselers %>%  
  rename(salary = income) %>%  
  mutate(age_months = age * 12)
```

Answer: No!

Chunk B

```
baselers <- baselers %>%  
  rename(salary = income) %>%  
  mutate(age_months = age * 12)
```

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.

```
df %>%  
  arrange(A, B)
```

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

```
df %>%  
  arrange(desc(A), B)
```

Sort by arm.

```
combined_df %>%  
  arrange(arm)    # Sort by arm
```

```
## # 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     4    34     1 placebo    100    105  
## 3     2    65     2 drug      143    140  
## 4     3    57     2 drug        NA     NA  
## 5     5    45     2 drug        NA     NA
```

arrange()

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

```
df %>%  
  arrange(A, B)
```

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

```
df %>%  
  arrange(desc(A), B)
```

Sort by arm and then age.

```
combined_df %>%  
  arrange(arm, age)  # Sort by arm then age
```

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


slice()

Use `slice()` to **select rows** (and remove others) by row number.

Use functions like `c()`, `a:b` and `seq()` to create row numbers

```
# Specific numbers  
c(2, 6, 10)
```

```
## [1] 2 6 10
```

```
# Integers from 0 to 5  
0:5
```

```
## [1] 0 1 2 3 4 5
```

Select rows 3 and 5.

```
# Rows 3 and 5 only  
combined_df %>%  
  slice(c(3, 5))
```

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

slice()

Use `slice()` to **select rows** (and remove others) by row number.

Use functions like `c()`, `a:b` and `seq()` to create row numbers

```
# Specific numbers  
c(2, 6, 10)
```

```
## [1] 2 6 10
```

```
# Integers from 0 to 5  
0:5
```

```
## [1] 0 1 2 3 4 5
```

Select rows 1 through 5.

```
# First 5 rows  
combined_df %>%  
  slice(1:5)
```

```
## # 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.

```
# Filter patients older than 30
combined_df %>%
  filter(age > 30)
```

```
## # 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 %>%
  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 A, B
df %>%
  select(A, B)
```

Remove columns with `-`.

```
# Select everything BUT A
df %>%
  select(-A)
```

Select columns `id` and `arm`

```
combined_df %>%
  select(id, arm) # Select id and arm columns
```

```
## # A tibble: 5 x 2
##       id    arm
##   <dbl> <dbl>
## 1     1     1
## 2     2     2
## 3     3     2
## 4     4     1
## 5     5     2
```

select()

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

```
# Select columns A, B
df %>%
  select(A, B)
```

Remove columns with -.

```
# Select everything BUT A
df %>%
  select(-A)
```

Select everything **id**

```
combined_df %>%
  select(-id) # Everything BUT id
```

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

Quiz 4

Here is part of the baselers dataframe

```
baselers %>%  
  select(id, sex, age, height) %>%  
  slice(1:5)
```

```
## # A tibble: 5 x 4  
##       id sex    age height  
##   <int> <chr> <int> <dbl>  
## 1     1 male    44   174.  
## 2     2 male    65   180.  
## 3     3 female  31   168.  
## 4     4 male    27   209  
## 5     5 male    24   177.
```

How do I calculate the following table of the top 5 tallest Baselers?

```
## # A tibble: 5 x 3  
##       id height sex  
##   <int>   <dbl> <chr>  
## 1   9676   219. male  
## 2   5623   213. male  
## 3   7214   213. male  
## 4   7059   212. male  
## 5   9538   210. male
```

Quiz 4

Here is part of the baselers dataframe

```
baselers %>%  
  select(id, sex, age, height) %>%  
  slice(1:5)
```

```
## # A tibble: 5 x 4  
##       id sex    age height  
##   <int> <chr> <int> <dbl>  
## 1     1 male    44   174.  
## 2     2 male    65   180.  
## 3     3 female  31   168.  
## 4     4 male    27   209  
## 5     5 male    24   177.
```

How do I calculate the following table of the top 5 tallest Baselers?

```
baselers %>%  
  arrange(desc(height)) %>%  
  select(id, height, sex) %>%  
  slice(1:5)
```

```
## # A tibble: 5 x 3  
##       id height sex  
##   <int> <dbl> <chr>  
## 1  9676   219. male  
## 2  5623   213. male  
## 3  7214   213. male  
## 4  7059   212. male  
## 5  9538   210. male
```


Quiz 5

Here is part of the baselers dataframe

```
baselers %>%  
  select(id, sex, age, height) %>%  
  slice(1:5)
```

```
## # A tibble: 5 x 4  
##       id sex    age height  
##   <int> <chr> <int> <dbl>  
## 1     1 male    44   174.  
## 2     2 male    65   180.  
## 3     3 female   31   168.  
## 4     4 male    27   209  
## 5     5 male    24   177.
```

How do I calculate the following table of the top 5 tallest Baselers?

```
## # A tibble: 5 x 3  
##       id height sex  
##   <int>   <dbl> <chr>  
## 1   6936   198. female  
## 2   8450   196. female  
## 3    385   196. female  
## 4   3203   195. female  
## 5   4392   194. female
```

Quiz 5

Here is part of the baselers dataframe

```
baselers %>%  
  select(id, sex, age, height) %>%  
  slice(1:5)
```

```
## # A tibble: 5 x 4  
##       id sex    age height  
##   <int> <chr> <int> <dbl>  
## 1     1 male    44   174.  
## 2     2 male    65   180.  
## 3     3 female   31   168.  
## 4     4 male    27   209  
## 5     5 male    24   177.
```

How do I calculate the following table of the top 5 tallest Baselers?

```
baselers %>%  
  filter(sex == "female") %>%  
  arrange(desc(height)) %>%  
  select(id, height, sex) %>%  
  slice(1:5)
```

```
## # A tibble: 5 x 3  
##       id height sex  
##   <int> <dbl> <chr>  
## 1  6936   198. female  
## 2  8450   196. female  
## 3   385   196. female  
## 4  3203   195. female  
## 5  4392   194. female
```

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 |
|-----------------------|--|
| <code>gather()</code> | Move data from 'wide' to 'long' format |
| <code>spread()</code> | 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 <- baselers %>%

# 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)
```

Practical

[Link to practical](#)