How to enter the tidyverse of madness

Matthew Suderman

Senior Lecturer in Epigenetic Epidemiology

MRC Integrative Epidemiology Unit

University of BRISTOL

What's wrong with R?

R is old (circa 1993).

It was originally designed for doing **statistics** with relatively small, simple datasets.

Since then, it has become one of the most popular tools for the new discipline called **data science**.

> "Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from noisy, structured and unstructured data, and apply knowledge and actionable insights from data across a broad range of application domains." https://en.wikipedia.org/wiki/Data_science

Shortcomings of R for data science

1. Lacks **consistency**

Shortcomings of R for data science

1. Lacks **consistency**
2. Often **suprising**

Shortcomings of R for data science

1. Lacks **consistency**
2. Often **suprising**
3. Tends to be **slow** and **memory-intensive**

Shortcomings of R for data science

1. Lacks **consistency**
2. Often **suprising**
3. Tends to be **slow** and **memory-intensive**
4. Built around **vectors and matrices**

Shortcomings of R for data science

1. Lacks **consistency**
2. Often **suprising**
3. Tends to be **slow** and **memory-intensive**
4. Built around **vectors and matrices**
5. Functional language can be **awkward** to use

1. Lacks consistency

function naming

```
names, colnames
row.names, rownames
rowSums, rowsum
rowMeans, (no parallel rowmean exists)
browseURL, contrib.url, fixup.package.UR
package.contents, packageStatus
getMethod, getS3method
read.csv and write.csv, load and save, r
Sys.time, system.time
```

[https://r4stats.com/articles/why-r-is-hard-to-learn/]

multiple ways to do things

e.g. two different object-oriented systems: S3 and S4 systems

missing values

```
> x <- c(1,2,3,4,NA)
> y <- c(1.1,2.2,3.3,4.4,5.5)

## missing values not okay
> quantile(x)
Error in quantile.default(x) :
  missing values not allowed if 'na.rm'

## missing values okay
> fit <- lm(y ~ x)
> length(x)
[1] 5
> length(residuals(fit))
[1] 4
```

2. Often surprising

R often converts between data types behind the scenes, e.g. characters to factors, matrices to vectors, numeric to characters

```
> x <- matrix(1:12,ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

2. Often surprising

R often converts between data types behind the scenes, e.g. characters to factors, matrices to vectors, numeric to characters

```
> x <- matrix(1:12,ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
> class(x[,1:2])
[1] "matrix" "array"
```

2. Often surprising

R often converts between data types behind the scenes, e.g. characters to factors, matrices to vectors, numeric to characters

```
> x <- matrix(1:12,ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
> class(x[,1:2])
[1] "matrix" "array"
```

```
> class(x[,3]) ## what is this?
```

2. Often surprising

R often converts between data types behind the scenes, e.g. characters to factors, matrices to vectors, numeric to characters

```
> x <- matrix(1:12,ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
> class(x[,1:2])
[1] "matrix" "array"
```

```
> class(x[,3]) ## what is this?
```

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

```
> x <- matrix(1:12,ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
> x <- matrix(1:12,ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
> x[1,1]
1
```

```
> x <- matrix(1:12,ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
> x[1,1]
1
```

```
> x[1,3] <- "a"
```

```
> x <- matrix(1:12,ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
> x[1,1]
1
```

```
> x[1,3] <- "a"
```

```
> x[1,1] + 1 ## what is this equal to?
```

```
> x <- matrix(1:12,ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
> x[1,1]
1
```

```
> x[1,3] <- "a"
```

```
> x[1,1] + 1 ## what is this equal to?
```

```
Error in x[1, 1] + 1 : non-numeric argument to binary operator
```

```
> x <- matrix(1:12,ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
> x[1,1]
1
```

```
> x[1,3] <- "a"
```

```
> x[1,1] + 1 ## what is this equal to?
```

```
Error in x[1, 1] + 1 : non-numeric argument to binary operator
```

```
> class(x[1,1])
[1] "character"
```

```
> x <- matrix(1:12,ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
> x[1,1]
1
```

```
> x[1,3] <- "a"
```

```
> x[1,1] + 1 ## what is this equal to?
```

```
Error in x[1, 1] + 1 : non-numeric argument to binary operator
```

```
> class(x[1,1])
[1] "character"
```

```
> x[1,1]
[1] "1"
```

3. Tends to be **slow** and **memory-intensive**

R was designed for small datasets, so it tends to do things

- the *simple* way

rather than

- the *efficient* way.

3. Tends to be **slow** and **memory-intensive**

R was designed for small datasets, so it tends to do things

- the *simple* way

rather than

- the *efficient* way.

e.g. it is designed to **slurp** data

Slurping means loading entire datasets into memory before analysing them.

This is not efficient when the analysis really only needs to see a table row-by-row.

But, this it simpler to just load (slurp) the entire dataset in one simple step.

4. Built around **vectors and matrices**

... whereas data science analyses **data frames** like this:

participant	age	sex	arm
1	23.58224	F	arm3
2	24.69102	M	arm4
3	25.42567	F	arm2
4	24.96019	F	arm2
5	23.35406	M	arm4

Each column has its own data type.

4. Built around **vectors and matrices**

... whereas data science analyses **data frames** like this:

participant	age	sex	arm
1	23.58224	F	arm3
2	24.69102	M	arm4
3	25.42567	F	arm2
4	24.96019	F	arm2
5	23.35406	M	arm4

Each column has its own data type.

> Although R implements data frames, most functions are designed for vectors and matrices.

5. Functional language that can be **awkward** to use

e.g. even if you know what all the functions do, this expression is difficult to understand!

```
summarize(group_by(left_join(filter(visits, wave=="wave2"), participants, by="part
age=mean(age), bmi=mean(bmi), blink=mean(blink))
```

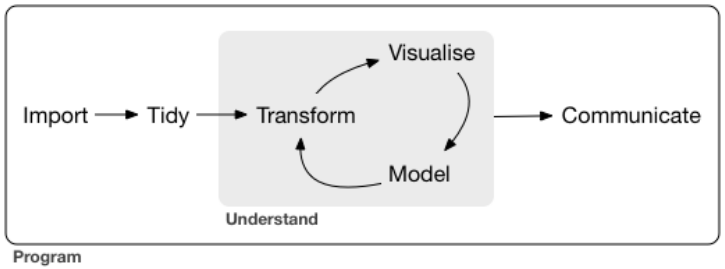
tidyverse

Tidyverse is a collection of R packages that aim to make the analysis of large datasets more convenient.

tidyverse

Tidyverse is a collection of R packages that aim to make the analysis of large datasets more convenient.

Here is intended workflow in the tidyverse:



- Data is **import**ed and made **tidy** so it can be analysed
- Analysis involves **transform**ing the data
- Relationships in the data are **visualised**
- **Models** are applied to test relationships
- Finally, conclusions are **communicate**d

tidyverse book

https://r4ds.had.co.nz/

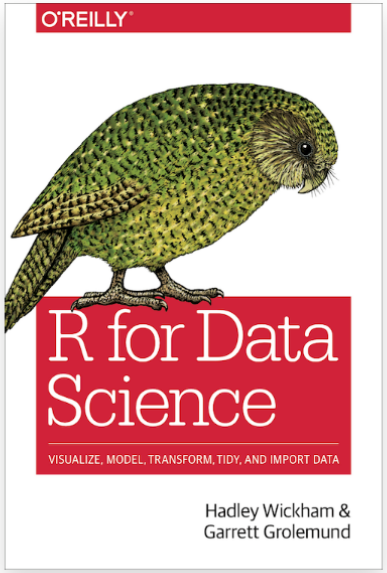
R for Data Science

Search

Table of contents

Welcome

This is the website for **"R for Data Science"**. This book will teach you how to do data science with R: You'll learn how to get your data into R, get it into the most useful structure, transform it, visualise it and model it. In this book, you will find a practicum of skills for data science. Just as a chemist learns how to clean test tubes and stock a lab, you'll learn how to clean data and draw plots—and many other things besides. These are the skills that allow data science to happen, and here you will find the best practices for doing each of these things with R. You'll learn how to use the grammar of graphics, literate programming, and reproducible research to save time. You'll also learn how to manage



Using tidyverse packages

Installing all tidyverse packages.

```
install.packages("tidyverse")
```

Loading all tidyverse packages.

```
library(tidyverse)
```

Our dataset

In this session, we will explore the tidyverse by performing a basic data analysis of a simulated dataset from a randomized control trial.

Our dataset

In this session, we will explore the tidyverse by performing a basic data analysis of a simulated dataset from a randomized control trial.

Background (Pretend) studies have shown that blink rate is positively associated with disengaged attention. A drug has been developed that is known to reduce blink rate.

Aim Determine if the drug will reduce attention deficits by reducing blink rate.

Methods

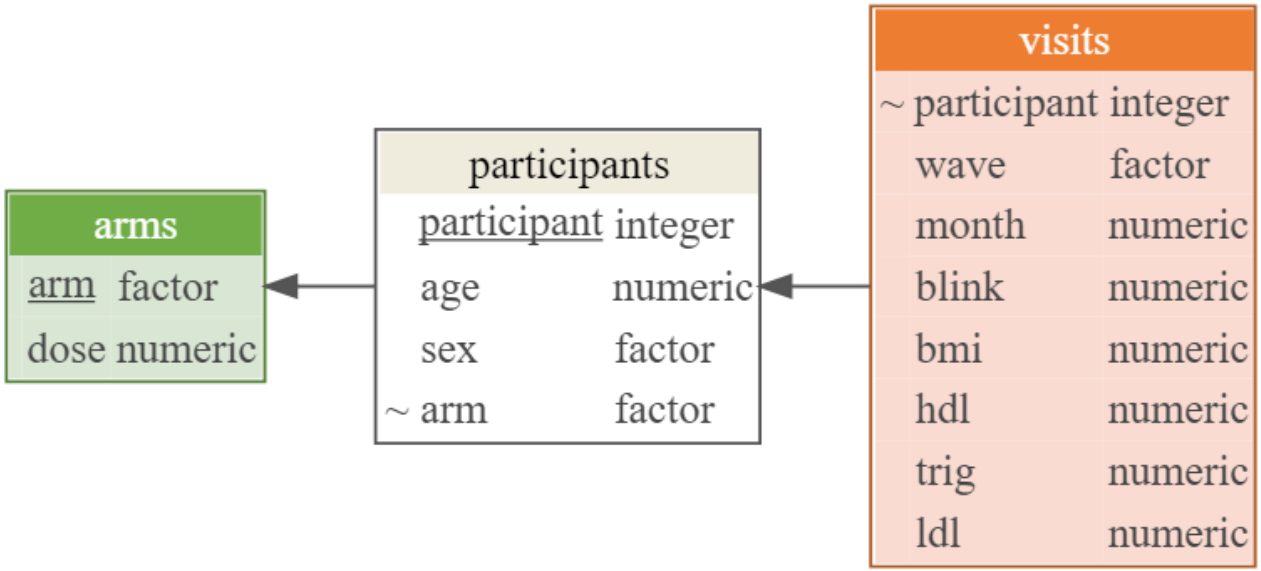
The were 4 arms:

controls (arm 1), low dose (arm 2), medium (arm 3), high (arm 4)

Each arm had 100 randomly selected individuals.

Data was collected in 3 waves:

Dataset schema



Import the dataset

The dataset is available as CSV files.

These can be imported into R using tidyverse function `readr::read_csv()`.

```
data.dir <- "https://raw.githubusercontent.com/perishky/perishky.github.io/master/

arms <- read_csv(file.path(data.dir, "arms.csv"))
participants <- read_csv(file.path(data.dir, "participants.csv"))
visits <- read_csv(file.path(data.dir, "visits.csv"))
```

Import the dataset

The dataset is available as CSV files.

These can be imported into R using tidyverse function `readr::read_csv()`.

```
data.dir <- "https://raw.githubusercontent.com/perishky/perishky.github.io/master/

arms <- read_csv(file.path(data.dir, "arms.csv"))
participants <- read_csv(file.path(data.dir, "participants.csv"))
visits <- read_csv(file.path(data.dir, "visits.csv"))
```

```
> arms <- read_csv("arms.csv")
Rows: 4 Columns: 2
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (1): arm
dbl (1): dose

> arms
# A tibble: 4 × 2
  arm    dose
  <chr> <dbl>
1 arm3      2
2 arm4      4
3 arm2      1
```

Why use `read_csv` rather than `read.csv`?

Why use `read_csv` rather than `read.csv`?

Speed `read_csv` is typically about 10x faster. If this isn't fast enough, `data.table::fread` is even faster!

Why use `read_csv` rather than `read.csv`?

Speed `read_csv` is typically about 10x faster. If this isn't fast enough, `data.table::fread` is even faster!

Better communication `read_csv` provides more information about what it has imported into R and clearer error messages if the import failed via the `problems()` function.

```
> arms <- read_csv("arms.csv")
Rows: 4 Columns: 2
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (1): arm
dbl (1): dose
```

Why use `read_csv` rather than `read.csv`?

Speed `read_csv` is typically about 10x faster. If this isn't fast enough, `data.table::fread` is even faster!

Better communication `read_csv` provides more information about what it has imported into R and clearer error messages if the import failed via the `problems()` function.

```
> arms <- read_csv("arms.csv")
Rows: 4 Columns: 2
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (1): arm
dbl (1): dose
```

Tidy format `read_csv` creates tibbles rather than data frames. (we'll discuss tibbles vs data frames on the next slide)

```
> arms
# A tibble: 4 × 2
  arm    dose
  <chr> <dbl>
1 arm3      2
```

Tidy

Tidy data is stored in tables called 'tibbles'.

1. "Each variable must have its own column."
2. "Each observation must have its own row."
3. "Each value must have its own cell."

Tidy

Tidy data is stored in tables called 'tibbles'.

1. "Each variable must have its own column."
2. "Each observation must have its own row."
3. "Each value must have its own cell."

   Why use a tibble rather than a data frame? (after all, tibbles are data frames)

Tidy

Tidy data is stored in tables called 'tibbles'.

1. "Each variable must have its own column."
2. "Each observation must have its own row."
3. "Each value must have its own cell."

   Why use a tibble rather than a data frame? (after all, tibbles are data frames)

   Predictability `tibble` never changes the types of inputs
   e.g. character strings are never converted to factors. It never changes column names. It never adds row names.

Tidy

Tidy data is stored in tables called 'tibbles'.

1. "Each variable must have its own column."
2. "Each observation must have its own row."
3. "Each value must have its own cell."

   Why use a tibble rather than a data frame? (after all, tibbles are data frames)

   Predictability `tibble` never changes the types of inputs
   e.g. character strings are never converted to factors. It never changes column names. It never adds row names.

   Convenience Whereas `tibble` can create data frames column-by-column, there is a `tribble` function to create data frames row-by-row.

Tidy

Tidy data is stored in tables called 'tibbles'.

1. "Each variable must have its own column."
2. "Each observation must have its own row."
3. "Each value must have its own cell."

Why use a tibble rather than a data frame? (after all, tibbles are data frames)

Predictability `tibble` never changes the types of inputs
e.g. character strings are never converted to factors. It never changes column names. It never adds row names.

Convenience Whereas `tibble` can create data frames column-by-column, there is a `tribble` function to create data frames row-by-row.

Pretty printing Tibbles are nicer to look at!
Typing the name of a data frame in R and pressing enter will print the entire data frame to the screen, regardless of its length or width! Tibbles will only show the first few rows and columns along with the dimensions of the data frame and data type of each column.

Creating tibbles

Tibbles can be created by column (just like a data frame)

```
tibble(
  x=c("a","b"),
  y=c(3,1),
  z=c(pi,2*pi)
)
```

Creating tibbles

Tibbles can be created by column (just like a data frame)

```
tibble(
  x=c("a","b"),
  y=c(3,1),
  z=c(pi,2*pi)
)
```

or by row like they'd be written in a data file

```
tribble(
  ~x, ~y, ~z,
  #--|--|----
  "a", 3, pi,
  "b", 1,2*pi
)
```

Answering questions

How many participants are in each study arm?

Answering questions

How many participants are in each study arm?

We'll use the `participants` tibble.

```
> participants
# A tibble: 400 × 4
   participant   age sex   arm
         <dbl> <dbl> <chr> <chr>
 1           1  23.6 F     arm3
 2           2  24.7 M     arm4
 3           3  25.4 F     arm2
 4           4  25.0 F     arm2
 5           5  23.4 M     arm4
 6           6  25.9 M     arm2
 7           7  26.5 M     arm4
 8           8  26.6 F     arm1
 9           9  25.3 F     arm2
10          10  25.3 F     arm3
# … with 390 more rows
```

Answering questions

How many participants are in each study arm?

We'll use the `participants` tibble.

```
> participants
# A tibble: 400 × 4
   participant   age sex   arm
         <dbl> <dbl> <chr> <chr>
 1           1  23.6 F     arm3
 2           2  24.7 M     arm4
 3           3  25.4 F     arm2
 4           4  25.0 F     arm2
 5           5  23.4 M     arm4
 6           6  25.9 M     arm2
 7           7  26.5 M     arm4
 8           8  26.6 F     arm1
 9           9  25.3 F     arm2
10          10  25.3 F     arm3
# … with 390 more rows
```

We'll use the `count()` function to count participants in each arm.

```
> count(participants, arm)
# A tibble: 4 × 2
   arm       n
 <chr>   <int>
1 arm1      100
2 arm2      100
3 arm3      100
4 arm4      100
```

`count()` is very similar to the `table()` function in base R.

Challenge!

See if you can count the number of female participants in each arm.

Challenge!

See if you can count the number of female participants in each arm.

```
> count(participants, arm, sex)
# A tibble: 8 × 3
  arm   sex       n
  <chr> <chr> <int>
1 arm1  F        48
2 arm1  M        52
3 arm2  F        43
4 arm2  M        57
5 arm3  F        50
6 arm3  M        50
7 arm4  F        46
8 arm4  M        54
```

Filtering

That last output gives more information than we wanted. To focus on just females, we'll need to filter out information about males.

To do this, we use the `filter()` function.

```
> counts <- count(participants, arm, sex)
> filter(counts, sex == "F")
# A tibble: 4 × 3
  arm   sex       n
  <chr> <chr> <int>
1 arm1  F        48
2 arm2  F        43
3 arm3  F        50
4 arm4  F        46
```

Challenge!

See if you can obtain the same result by

1. first applying the `filter` function to `participants` to remove males
2. and then applying the `count` function to the result.

Challenge!

See if you can obtain the same result by

1. first applying the `filter` function to `participants` to remove males
2. and then applying the `count` function to the result.

```
> females <- filter(participants, sex=="F")
> count(females, arm, sex)
# A tibble: 4 × 3
  arm   sex       n
  <chr> <chr> <int>
1 arm1  F        48
2 arm2  F        43
3 arm3  F        50
4 arm4  F        46
```

Intermediate variables

For longer queries, it can get a little tedious creating variable names for each step.

In the queries above, we had to create `counts` and `females` even though we aren't directly interested in those variables, e.g.

```
counts <- count(participants, arm, sex)
filter(counts, sex == "F")
```

Intermediate variables

For longer queries, it can get a little tedious creating variable names for each step.

In the queries above, we had to create `counts` and `females` even though we aren't directly interested in those variables, e.g.

```
counts <- count(participants, arm, sex)
filter(counts, sex == "F")
```

One solution is called **function composition**.

```
filter(count(participants, arm, sex), sex=="F")
```

Intermediate variables

For longer queries, it can get a little tedious creating variable names for each step.

In the queries above, we had to create `counts` and `females` even though we aren't directly interested in those variables, e.g.

```
counts <- count(participants, arm, sex)
filter(counts, sex == "F")
```

One solution is called **function composition**.

```
filter(count(participants, arm, sex), sex=="F")
```

Challenge!

See if you can obtain the same result with `count` as the outer function, i.e. `count(filter(...), ...)`

Intermediate variables

For longer queries, it can get a little tedious creating variable names for each step.

In the queries above, we had to create `counts` and `females` even though we aren't directly interested in those variables, e.g.

```
counts <- count(participants, arm, sex)
filter(counts, sex == "F")
```

One solution is called **function composition**.

```
filter(count(participants, arm, sex), sex=="F")
```

Challenge!

See if you can obtain the same result with `count` as the outer function, i.e. `count(filter(...), ...)`

```
count(filter(participants, sex=="F"), arm, sex)
```

Function composition gets extreme

Although function composition avoids having to name intermediate variables, they can make the code difficult to read.

For example, the following command summarizes information about each study arm (we'll discuss the functions used here later).

```
summarize(group_by(left_join(filter(visits, wave=="wave2"),
  participants, by="participant"),arm,sex),n=n(), age=mean(age), bmi=mean(bmi), bl
```

Function composition gets extreme

Although function composition avoids having to name intermediate variables, they can make the code difficult to read.

For example, the following command summarizes information about each study arm (we'll discuss the functions used here later).

```
summarize(group_by(left_join(filter(visits, wave=="wave2"),
  participants, by="participant"),arm,sex),n=n(), age=mean(age), bmi=mean(bmi), bl
```

To be fair, we can split it across lines to make it a little easier to read.

```
summarize(
  group_by(
   left_join(
     filter(visits, wave=="wave2"),
     participants,
     by="participant"),
    arm,
    sex),
  n=n(),
  age=mean(age),
  bmi=mean(bmi),
  blink=mean(blink))
```

Pipes

The "pipe" operator (`%>%`) was invented to solve this problem.

This command ...

```
filter(count(participants, arm, sex), sex=="F")
```

Pipes

The "pipe" operator (`%>%`) was invented to solve this problem.

This command ...

```
filter(count(participants, arm, sex), sex=="F")
```

... can be written like this

```
participants %>% filter(sex=="F") %>% count(arm, sex)
```

Pipes

The "pipe" operator (`%>%`) was invented to solve this problem.

This command ...

```
filter(count(participants, arm, sex), sex=="F")
```

... can be written like this

```
participants %>% filter(sex=="F") %>% count(arm, sex)
```

... or even better, split across multiple lines

```
participants %>%
  filter(sex=="F") %>%
  count(arm, sex)
```

i.e.

- we 'pipe' `participants` to `filter`,
- and then the output of `filter` to `count`

Challenge!

See if you can rewrite the following using pipes:

```
filter(count(participants,arm,sex), sex=="F")
```

Challenge!

See if you can rewrite the following using pipes:

```
filter(count(participants,arm,sex), sex=="F")
```

```
participants %>%
  count(arm,sex) %>%
  filter(sex=="F")
```

Challenge!

Now see if you can rewrite this command using pipes:

```
summarize(
  group_by(
   left_join(
     filter(visits, wave=="wave2"),
     participants,
     by="participant"),
    arm,
    sex),
  n=n(),
  age=mean(age),
  bmi=mean(bmi),
  blink=mean(blink))
```

Challenge!

Now see if you can rewrite this command using pipes:

```
summarize(
  group_by(
   left_join(
     filter(visits, wave=="wave2"),
     participants,
     by="participant"),
    arm,
    sex),
  n=n(),
  age=mean(age),
  bmi=mean(bmi),
  blink=mean(blink))
```

```
visits %>%
  filter(wave=="wave2") %>%
  left_join(participants, by="participant") %>%
  group_by(arm,sex) %>%
  summarize(n=n(),age=mean(age), bmi=mean(bmi), blink=mean(blink))
```

Calculating data summaries

The `summarise(<tibble>,<summary1>,<summary2>,...)` summarizes the columns of a tibble.

For example, we can use it to calculate

1. the average participant age, and
2. how many of them are females.

Calculating data summaries

The `summarise(<tibble>,<summary1>,<summary2>,...)` summarizes the columns of a tibble.

For example, we can use it to calculate

1. the average participant age, and
2. how many of them are females.

```
> summarise(participants, age=median(age), nfemales=sum(sex=="F"))
# A tibble: 1 × 2
    age nfemales
  <dbl>    <int>
1  25.0      187
```

Calculating data summaries

The `summarise(<tibble>,<summary1>,<summary2>,...)` summarizes the columns of a tibble.

For example, we can use it to calculate

1. the average participant age, and
2. how many of them are females.

```
> summarise(participants, age=median(age), nfemales=sum(sex=="F"))
# A tibble: 1 × 2
    age nfemales
  <dbl>    <int>
1  25.0      187
```

Here is what it looks like using pipes.

```
participants %>%
  summarise(age=median(age), nfemales=sum(sex=="F"))
```

Calculating summaries of subsets

The previous example above provided summaries across *all* rows.

It is also possible to summarize subsets of rows using `group_by(<tibble>, <groupby1>,<groupby2>,...)`.

For example, we can calculate, *for each arm of the study*

1. the average participant age, and
2. the number of males and females.

Calculating summaries of subsets

The previous example above provided summaries across *all* rows.

It is also possible to summarize subsets of rows using `group_by(<tibble>, <groupby1>,<groupby2>,...)`.

For example, we can calculate, *for each arm of the study*

1. the average participant age, and
2. the number of males and females.

```
> participants %>%
  group_by(arm,sex) %>%
  summarise(n=n(), age=mean(age))

# A tibble: 8 × 4
# Groups:   arm [4]
  arm   sex       n   age
  <chr> <chr> <int> <dbl>
1 arm1  F        48  25.1
2 arm1  M        52  25.1
3 arm2  F        43  25.1
4 arm2  M        57  24.8
5 arm3  F        50  25.1
6 arm3  M        50  24.8
7 arm4  F        46  24.9
```

Calculating summaries of subsets

The previous example above provided summaries across *all* rows.

It is also possible to summarize subsets of rows using `group_by(<tibble>, <groupby1>,<groupby2>,...)`.

For example, we can calculate, *for each arm of the study*

1. the average participant age, and
2. the number of males and females.

```
> participants %>%
  group_by(arm,sex) %>%
  summarise(n=n(), age=m

# A tibble: 8 × 4
# Groups:   arm [4]
  arm   sex       n   ag
  <chr> <chr> <int> <db
1 arm1  F        48  25.
2 arm1  M        52  25.
3 arm2  F        43  25.
4 arm2  M        57  24.
5 arm3  F        50  25.
6 arm3  M        50  24.
7 arm4  F        46  24.
```

Did you notice some **magic**?

1. `summarise()` magically knows when a tibble has come from `group_by()` and that it should summarize by the resulting groups.
2. `n()` function magically knows about the tibble subsets being processed by `summarise`.

Challenge!

Can you think of an alternative to the mysterious `n()` function? (*this is a tricky question*)

```
participants %>%
  group_by(arm,sex) %>%
  summarise(n=n(), age=mean(age))
```

Challenge!

Can you think of an alternative to the mysterious `n()` function? (*this is a tricky question*)

```
participants %>%
  group_by(arm,sex) %>%
  summarise(n=n(), age=mean(age))
```

Hint: Use the `length` function ...

Challenge!

Can you think of an alternative to the mysterious `n()` function? (*this is a tricky question*)

```
participants %>%
  group_by(arm,sex) %>%
  summarise(n=n(), age=mean(age))
```

Hint: Use the `length` function ...

```
participants %>%
  group_by(arm,sex) %>%
  summarise(n=length(age), age=mean(age))
```

Merging datasets

We'd like to augment the summary with measurements from the first visit.

This information is in the `visits` dataset.

To do this, we'll first need to:

1. filter `visits` to get just the first visit (there were 3 visits)
2. merge the result with `participants`
3. group the result by arm and by sex
4. calculate the mean of age, BMI and blink rate

Merging datasets

We'd like to augment the summary with measurements from the first visit.

This information is in the `visits` dataset.

To do this, we'll first need to:

1. filter `visits` to get just the first visit (there were 3 visits)
2. merge the result with `participants`
3. group the result by arm and by sex
4. calculate the mean of age, BMI and blink rate

We've seen how to do each of these steps except step 2, merging.

```
visits %>%
  filter(wave=="wave0") %>%
  ????????? merge with participants ????????? %>%
  group_by(arm,sex) %>%
  summarise(n=n(), age=mean(age), bmi=mean(bmi), blink=mean(blink))
```

Merging datasets

We'd like to augment the summary with measurements from the first visit.

This information is in the `visits` dataset.

To do this, we'll first need to:

1. filter `visits` to get just the first visit (there were 3 visits)
2. merge the result with `participants`
3. group the result by arm and by sex
4. calculate the mean of age, BMI and blink rate

We've seen how to do each of these steps except step 2, merging.

```
visits %>%
  filter(wave=="wave0") %>%
  ????????? merge with participants ????????? %>%
  group_by(arm,sex) %>%
  summarise(n=n(), age=mean(age), bmi=mean(bmi), blink=mean(blink))
```

In base R, we'd use the function `merge()`.

For tibbles, we have four different functions: `left_join()`, `right_join()`, `inner_join()` and `full_join()`.

left_join()

To illustrate, we'll use a small version of `visits`.

```
> visits.s <- visits %>% filter(partici
> visits.s
# A tibble: 9 × 8
participant wave  month blink   bmi   h
      <dbl> <chr> <dbl> <dbl> <dbl> <db
1         1 wave0  0     17.6  15.4  1.
2         1 wave1  6.18  16.0  16.5  1.
3         1 wave2 10.4   15.6  14.9  1.
4         2 wave0  0     16.3  19.4  1.
5         2 wave1  5.50  14.5  20.6  1.
6         2 wave2 13.0   11.2  21.8  1.
7         3 wave0  0     18.6  22.0  1.
8         3 wave1  6.58  17.8  20.8  1.
9         3 wave2 13.5   17.3  23.1  1.
```

left_join()

To illustrate, we'll use a small version of `visits`.

```
> visits.s <- visits %>% filter(partici
> visits.s
# A tibble: 9 × 8
participant wave  month blink   bmi   h
      <dbl> <chr> <dbl> <dbl> <dbl> <db
1         1 wave0  0     17.6  15.4  1.
2         1 wave1  6.18  16.0  16.5  1.
3         1 wave2 10.4   15.6  14.9  1.
4         2 wave0  0     16.3  19.4  1.
5         2 wave1  5.50  14.5  20.6  1.
6         2 wave2 13.0   11.2  21.8  1.
7         3 wave0  0     18.6  22.0  1.
8         3 wave1  6.58  17.8  20.8  1.
9         3 wave2 13.5   17.3  23.1  1.
```

Recall `participants`.

```
> participants
# A tibble: 400 × 4
   participant   age sex   arm
         <dbl> <dbl> <chr> <chr>
 1           1  23.6 F     arm3
 2           2  24.7 M     arm4
 3           3  25.4 F     arm2
 4           4  25.0 F     arm2
 5           5  23.4 M     arm4
 6           6  25.9 M     arm2
 7           7  26.5 M     arm4
 8           8  26.6 F     arm1
 9           9  25.3 F     arm2
10          10  25.3 F     arm3
# … with 390 more rows
```

left_join()

`left_join()` adds data from `participants` that match the three participants in `visits.s`.

```
> left_join(visits.s, participants, by="participant")
Joining, by = "participant"
# A tibble: 9 × 11
  participant wave  month blink   bmi   hdl  trig   ldl   age sex   arm
        <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr> <chr>
1           1 wave0  0     17.6  15.4  1.38  1.52  2.11  23.6 F     arm3
2           1 wave1  6.18  16.0  16.5  1.35  1.33  1.88  23.6 F     arm3
3           1 wave2 10.4   15.6  14.9  1.39  1.39  2.01  23.6 F     arm3
4           2 wave0  0     16.3  19.4  1.64  1.70  4.06  24.7 M     arm4
5           2 wave1  5.50  14.5  20.6  1.61  1.31  3.90  24.7 M     arm4
6           2 wave2 13.0   11.2  21.8  1.57  1.62  4.04  24.7 M     arm4
7           3 wave0  0     18.6  22.0  1.61  1.12  2.11  25.4 F     arm2
8           3 wave1  6.58  17.8  20.8  1.75  1.14  1.82  25.4 F     arm2
9           3 wave2 13.5   17.3  23.1  1.53  1.11  2.21  25.4 F     arm2
```

left_join()

`left_join()` adds data from `participants` that match the three participants in `visits.s`.

```
> left_join(visits.s, participants, by="participant")
Joining, by = "participant"
# A tibble: 9 × 11
  participant wave  month blink   bmi   hdl  trig   ldl   age sex   arm
        <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr> <chr>
1           1 wave0  0     17.6  15.4  1.38  1.52  2.11  23.6 F     arm3
2           1 wave1  6.18  16.0  16.5  1.35  1.33  1.88  23.6 F     arm3
3           1 wave2 10.4   15.6  14.9  1.39  1.39  2.01  23.6 F     arm3
4           2 wave0  0     16.3  19.4  1.64  1.70  4.06  24.7 M     arm4
5           2 wave1  5.50  14.5  20.6  1.61  1.31  3.90  24.7 M     arm4
6           2 wave2 13.0   11.2  21.8  1.57  1.62  4.04  24.7 M     arm4
7           3 wave0  0     18.6  22.0  1.61  1.12  2.11  25.4 F     arm2
8           3 wave1  6.58  17.8  20.8  1.75  1.14  1.82  25.4 F     arm2
9           3 wave2 13.5   17.3  23.1  1.53  1.11  2.21  25.4 F     arm2
```

Notice how the result is just `visits.s` but with some additional columns from `participants`.

Rows in `participants` that do not match are omitted.

Challenge!

What do you think happens if some participants in `visits` do not match a participant in `participants`?

```
> parts.s <- participants %>% filter(participant %in% 1:2)
> parts.s
# A tibble: 2 × 4
  participant   age sex   arm
        <dbl> <dbl> <chr> <chr>
1           1  23.6 F     arm3
2           2  24.7 M     arm4
```

Challenge!

What do you think happens if some participants in `visits` do not match a participant in `participants`?

```
> parts.s <- participants %>% filter(participant %in% 1:2)
> parts.s
# A tibble: 2 × 4
  participant   age sex   arm
        <dbl> <dbl> <chr> <chr>
1           1  23.6 F     arm3
2           2  24.7 M     arm4
```

```
> left_join(visits.s, parts.s, by="participant")
# A tibble: 9 × 11
  participant wave  month blink   bmi   hdl  trig   ldl   age sex   arm
        <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr> <chr>
1           1 wave0  0     17.6  15.4  1.38  1.52  2.11  23.6 F     arm3
2           1 wave1  6.18  16.0  16.5  1.35  1.33  1.88  23.6 F     arm3
3           1 wave2 10.4   15.6  14.9  1.39  1.39  2.01  23.6 F     arm3
4           2 wave0  0     16.3  19.4  1.64  1.70  4.06  24.7 M     arm4
5           2 wave1  5.50  14.5  20.6  1.61  1.31  3.90  24.7 M     arm4
6           2 wave2 13.0   11.2  21.8  1.57  1.62  4.04  24.7 M     arm4
7           3 wave0  0     18.6  22.0  1.61  1.12  2.11  NA   NA    NA
8           3 wave1  6.58  17.8  20.8  1.75  1.14  1.82  NA   NA    NA
9           3 wave2 13.5   17.3  23.1  1.53  1.11  2.21  NA   NA    NA
```

Putting it altogether

Recall that we want to summarize information from all participants, grouped by age and sex.

```
visits %>%
  filter(wave=="wave0") %>%
  left_join(participants,by="participant") %>% ## merging step
  group_by(arm,sex) %>%
  summarise(n=n(), age=mean(age), bmi=mean(bmi), blink=mean(blink))
```

Putting it altogether

Recall that we want to summarize information from all participants, grouped by age and sex.

```
visits %>%
  filter(wave=="wave0") %>%
  left_join(participants,by="participant") %>% ## merging step
  group_by(arm,sex) %>%
  summarise(n=n(), age=mean(age), bmi=mean(bmi), blink=mean(blink))
```

```
# A tibble: 8 × 6
# Groups:   arm [4]
  arm   sex       n   age   bmi blink
  <chr> <chr> <int> <dbl> <dbl> <dbl>
1 arm1  F        48  25.1  21.4  17.7
2 arm1  M        52  25.1  23.9  17.5
3 arm2  F        43  25.1  21.3  17.6
4 arm2  M        57  24.8  23.4  17.5
5 arm3  F        50  25.1  21.0  17.7
6 arm3  M        50  24.8  23.8  17.6
7 arm4  F        46  24.9  21.4  17.2
8 arm4  M        54  25.1  22.9  17.6
```

Challenge!

The previous output wasn't interesting because we're looking at the data before any treatment.

Modify the command to see the same data from the final visit.

Challenge!

The previous output wasn't interesting because we're looking at the data before any treatment.

Modify the command to see the same data from the final visit.

```
visits %>%
  filter(wave=="wave2") %>%
  left_join(participants,by="participant") %>%
  group_by(arm,sex) %>%
  summarise(n=n(), age=mean(age), bmi=mean(bmi), blink=mean(blink))
```

Challenge!

The previous output wasn't interesting because we're looking at the data before any treatment.

Modify the command to see the same data from the final visit.

```
visits %>%
  filter(wave=="wave2") %>%
  left_join(participants,by="participant") %>%
  group_by(arm,sex) %>%
  summarise(n=n(), age=mean(age), bmi=mean(bmi), blink=mean(blink))
```

```
# A tibble: 8 × 6
# Groups:   arm [4]
  arm   sex       n   age   bmi blink
  <chr> <chr> <int> <dbl> <dbl> <dbl>
1 arm1  F        48  25.1  21.6  17.7
2 arm1  M        52  25.1  24.0  17.5
3 arm2  F        43  25.1  21.6  16.5
4 arm2  M        57  24.8  24.0  16.3
5 arm3  F        50  25.1  22.1  15.3
6 arm3  M        50  24.8  25.1  15.2
7 arm4  F        46  24.9  23.7  12.6
8 arm4  M        54  25.1  25.2  13.0
```

Merging multiple tables

To remember the treatment within each study arm, we'll add 'dose' to the output found in the `arms` tibble.

Merging multiple tables

To remember the treatment within each study arm, we'll add 'dose' to the output found in the `arms` tibble.

```
query <- visits %>%
  filter(wave=="wave2") %>%
  left_join(participants,by="participant") %>%
  left_join(arms,by="arm") %>%
  group_by(arm,sex) %>%
  summarise(dose=dose[1],n=n(), age=mean(age), bmi=mean(bmi), blink=mean(blink))
```

Merging multiple tables

To remember the treatment within each study arm, we'll add 'dose' to the output found in the `arms` tibble.

```
query <- visits %>%
  filter(wave=="wave2") %>%
  left_join(participants,by="participant") %>%
  left_join(arms,by="arm") %>%
  group_by(arm,sex) %>%
  summarise(dose=dose[1],n=n(), age=mean(age), bmi=mean(bmi), blink=mean(blink))
```

```
> query

# A tibble: 8 × 7
# Groups:   arm [4]
  arm   sex    dose     n   age   bmi blink
  <chr> <chr> <dbl> <int> <dbl> <dbl> <dbl>
1 arm1  F         0    48  25.1  21.6  17.7
2 arm1  M         0    52  25.1  24.0  17.5
3 arm2  F         1    43  25.1  21.6  16.5
4 arm2  M         1    57  24.8  24.0  16.3
5 arm3  F         2    50  25.1  22.1  15.3
6 arm3  M         2    50  24.8  25.1  15.2
7 arm4  F         4    46  24.9  23.7  12.6
8 arm4  M         4    54  25.1  25.2  13.0
```

Selecting columns

Having both 'arm' and 'dose' columns is a bit redundant, we just want to show 'dose'.

For this, we can use the `select()` function.

Selecting columns

Having both 'arm' and 'dose' columns is a bit redundant, we just want to show 'dose'.

For this, we can use the `select()` function.

```
> query %>%
  ungroup() %>%
  select(dose,sex,n,age,bmi,blink)

# A tibble: 8 × 6
   dose sex       n   age   bmi blink
  <dbl> <chr> <int> <dbl> <dbl> <dbl>
1     0 F        48  25.1  21.6  17.7
2     0 M        52  25.1  24.0  17.5
3     1 F        43  25.1  21.6  16.5
4     1 M        57  24.8  24.0  16.3
5     2 F        50  25.1  22.1  15.3
6     2 M        50  24.8  25.1  15.2
7     4 F        46  24.9  23.7  12.6
8     4 M        54  25.1  25.2  13.0
```

The `ungroup` command is necessary to turn 'off' grouping.

Omitting columns

It's a bit tedious to type all the column names when we just want to remove one. There is a way to do this.

Omitting columns

It's a bit tedious to type all the column names when we just want to remove one. There is a way to do this.

```
query <- query %>%
  ungroup() %>%
  select(-arm)
```

Omitting columns

It's a bit tedious to type all the column names when we just want to remove one. There is a way to do this.

```
query <- query %>%
  ungroup() %>%
  select(-arm)
```

```
> query
# A tibble: 8 × 6
  sex    dose     n   age   bmi blink
  <chr> <dbl> <int> <dbl> <dbl> <dbl>
1 F         0    48  25.1  21.6  17.7
2 M         0    52  25.1  24.0  17.5
3 F         1    43  25.1  21.6  16.5
4 M         1    57  24.8  24.0  16.3
5 F         2    50  25.1  22.1  15.3
6 M         2    50  24.8  25.1  15.2
7 F         4    46  24.9  23.7  12.6
8 M         4    54  25.1  25.2  13.0
```

Reordering rows

With male rows next to female rows, it is difficult to see if there were any treatment effects on BMI.

Reordering rows

With male rows next to female rows, it is difficult to see if there were any treatment effects on BMI.

We'll reorder/arrange the rows by sex and then by dose.

Before reordering ...

```
> query


# A tibble: 8 × 6
  sex    dose     n   age   bmi blink
  <chr> <dbl> <int> <dbl> <dbl> <dbl>
1 F         0    48  25.1  21.6  17.7
2 M         0    52  25.1  24.0  17.5
3 F         1    43  25.1  21.6  16.5
4 M         1    57  24.8  24.0  16.3
5 F         2    50  25.1  22.1  15.3
6 M         2    50  24.8  25.1  15.2
7 F         4    46  24.9  23.7  12.6
8 M         4    54  25.1  25.2  13.0
```

Reordering rows

With male rows next to female rows, it is difficult to see if there were any treatment effects on BMI.

We'll reorder/arrange the rows by sex and then by dose.

Before reordering ...

```
> query


# A tibble: 8 × 6
  sex    dose     n   age   bmi blink
  <chr> <dbl> <int> <dbl> <dbl> <dbl>
1 F         0    48  25.1  21.6  17.7
2 M         0    52  25.1  24.0  17.5
3 F         1    43  25.1  21.6  16.5
4 M         1    57  24.8  24.0  16.3
5 F         2    50  25.1  22.1  15.3
6 M         2    50  24.8  25.1  15.2
7 F         4    46  24.9  23.7  12.6
8 M         4    54  25.1  25.2  13.0
```

After reordering ...

```
> query %>%
  arrange(sex,dose)

# A tibble: 8 × 6
  sex    dose     n   age   bmi blink
  <chr> <dbl> <int> <dbl> <dbl> <dbl>
1 F         0    48  25.1  21.6  17.7
2 F         1    43  25.1  21.6  16.5
3 F         2    50  25.1  22.1  15.3
4 F         4    46  24.9  23.7  12.6
5 M         0    52  25.1  24.0  17.5
6 M         1    57  24.8  24.0  16.3
7 M         2    50  24.8  25.1  15.2
8 M         4    54  25.1  25.2  13.0
```

Now we can more easily see that BMI

Running statistical tests

We'd like to determine statistically if treatment actually increases BMI.

We can do this by fitting the following linear model:

$$BMI_{wave\ 2} \sim BMI_{wave\ 0} + age + sex + dose$$

Running statistical tests

We'd like to determine statistically if treatment actually increases BMI.

We can do this by fitting the following linear model:

$$BMI_{wave\ 2} \sim BMI_{wave\ 0} + age + sex + dose$$

For this, we'll need to prepare a tibble like this:

bmi.wave0	bmi.wave2	age	sex	dose
5.4	14.9	23.5	F	2
19.3	21.8	24.7	M	4
22.0	23.1	25.4	F	1
20.6	19.7	24.9	F	1
26.1	28.6	23.4	M	4
...				

Running statistical tests

Here are the steps:

Running statistical tests

Here are the steps:

1. Create a wave 0 subset of 'visits'

Running statistical tests

Here are the steps:

1. Create a wave 0 subset of 'visits'
2. Create a wave 2 subset of 'visits'

Running statistical tests

Here are the steps:

1. Create a wave 0 subset of 'visits'
2. Create a wave 2 subset of 'visits'
3. Merge these two so we have a data frame with bmi at wave 0 and wave 2

Running statistical tests

Here are the steps:

1. Create a wave 0 subset of 'visits'
2. Create a wave 2 subset of 'visits'
3. Merge these two so we have a data frame with bmi at wave 0 and wave 2
4. Merge this with 'participants' and 'arms' to add participant age, sex and dose

Running statistical tests

Here are the steps:

1. Create a wave 0 subset of 'visits'
2. Create a wave 2 subset of 'visits'
3. Merge these two so we have a data frame with bmi at wave 0 and wave 2
4. Merge this with 'participants' and 'arms' to add participant age, sex and dose
5. Fit the linear model and extract the coefficients and p-values

```
## 1. Create a wave 0 subset of 'visits'
```

```
## 1. Create a wave 0 subset of 'visits'
```

```
dat.wave0 <- visits %>% filter(wave=="wave0") %>% select(participant, bmi)
```

```
## 1. Create a wave 0 subset of 'visits'
```

```
dat.wave0 <- visits %>% filter(wave=="wave0") %>% select(participant, bmi)
```

```
## 2. Create a wave 2 subset of 'visits'
```

```
## 1. Create a wave 0 subset of 'visits'
```

```
dat.wave0 <- visits %>% filter(wave=="wave0") %>% select(participant, bmi)
```

```
## 2. Create a wave 2 subset of 'visits'
```

```
dat.wave2 <- visits %>% filter(wave=="wave2") %>% select(participant, bmi)
```

```
## 1. Create a wave 0 subset of 'visits'
```

```
dat.wave0 <- visits %>% filter(wave=="wave0") %>% select(participant, bmi)
```

```
## 2. Create a wave 2 subset of 'visits'
```

```
dat.wave2 <- visits %>% filter(wave=="wave2") %>% select(participant, bmi)
```

```
## 3. Merge these two so we have a data frame with bmi at wave 0 and wave 2
```

```
## 1. Create a wave 0 subset of 'visits'
```

```
dat.wave0 <- visits %>% filter(wave=="wave0") %>% select(participant, bmi)
```

```
## 2. Create a wave 2 subset of 'visits'
```

```
dat.wave2 <- visits %>% filter(wave=="wave2") %>% select(participant, bmi)
```

```
## 3. Merge these two so we have a data frame with bmi at wave 0 and wave 2
```

```
dat.wave0 %>%
  left_join(dat.wave2, by="participant", suffix=c(".wave0", ".wave2")) %>%
```

```
## 1. Create a wave 0 subset of 'visits'
```

```
dat.wave0 <- visits %>% filter(wave=="wave0") %>% select(participant, bmi)
```

```
## 2. Create a wave 2 subset of 'visits'
```

```
dat.wave2 <- visits %>% filter(wave=="wave2") %>% select(participant, bmi)
```

```
## 3. Merge these two so we have a data frame with bmi at wave 0 and wave 2
```

```
dat.wave0 %>%
  left_join(dat.wave2, by="participant", suffix=c(".wave0", ".wave2")) %>%
```

```
  ## 4. Merge this with 'participants' and 'arms' to add participant age, sex and
```

```
## 1. Create a wave 0 subset of 'visits'
```

```
dat.wave0 <- visits %>% filter(wave=="wave0") %>% select(participant, bmi)
```

```
## 2. Create a wave 2 subset of 'visits'
```

```
dat.wave2 <- visits %>% filter(wave=="wave2") %>% select(participant, bmi)
```

```
## 3. Merge these two so we have a data frame with bmi at wave 0 and wave 2
```

```
dat.wave0 %>%
  left_join(dat.wave2, by="participant", suffix=c(".wave0", ".wave2")) %>%
```

```
  ## 4. Merge this with 'participants' and 'arms' to add participant age, sex and
```

```
  left_join(participants, by="participant") %>%
  left_join(arms, by="arm") %>%
```

```
## 1. Create a wave 0 subset of 'visits'
```

```
dat.wave0 <- visits %>% filter(wave=="wave0") %>% select(participant, bmi)
```

```
## 2. Create a wave 2 subset of 'visits'
```

```
dat.wave2 <- visits %>% filter(wave=="wave2") %>% select(participant, bmi)
```

```
## 3. Merge these two so we have a data frame with bmi at wave 0 and wave 2
```

```
dat.wave0 %>%
  left_join(dat.wave2, by="participant", suffix=c(".wave0", ".wave2")) %>%
```

```
  ## 4. Merge this with 'participants' and 'arms' to add participant age, sex and
```

```
  left_join(participants, by="participant") %>%
  left_join(arms, by="arm") %>%
```

```
  ## 5. Fit the linear model and extract the coefficients and p-values
```

```
## 1. Create a wave 0 subset of 'visits'
```

```
dat.wave0 <- visits %>% filter(wave=="wave0") %>% select(participant, bmi)
```

```
## 2. Create a wave 2 subset of 'visits'
```

```
dat.wave2 <- visits %>% filter(wave=="wave2") %>% select(participant, bmi)
```

```
## 3. Merge these two so we have a data frame with bmi at wave 0 and wave 2
```

```
dat.wave0 %>%
  left_join(dat.wave2, by="participant", suffix=c(".wave0", ".wave2")) %>%
```

```
  ## 4. Merge this with 'participants' and 'arms' to add participant age, sex and
```

```
  left_join(participants, by="participant") %>%
  left_join(arms, by="arm") %>%
```

```
  ## 5. Fit the linear model and extract the coefficients and p-values
```

```
  lm(bmi.wave2 ~ bmi.wave0 + age + sex + dose, data=.) %>%
```

Wait, what does that mysterious '.' mean?

```
... %>%
lm(bmi.wave2 ~ bmi.wave0 + age + sex + dose, data=.) %>%
...
```

Wait, what does that mysterious '.' mean?

```
... %>%
lm(bmi.wave2 ~ bmi.wave0 + age + sex + dose, data=.) %>%
...
```

The '.' refers to the input coming from the the pipe ('%>%') to the `lm` function. We want the incoming data frame/tibble to be passed to `lm` via the `data` argument.

By default, pipe input is passed as the first argument to the function. Most tidyverse functions are written with this default in mind.

Running statistical tests

```
dat.wave0 <- visits %>% filter(wave=="wave0") %>% select(participant, bmi)
dat.wave2 <- visits %>% filter(wave=="wave2") %>% select(participant, bmi)
dat.wave0 %>%
  left_join(dat.wave2, by="participant", suffix=c(".wave0", ".wave2")) %>%
  left_join(participants, by="participant") %>%
  left_join(arms, by="arm") %>%
  lm(bmi.wave2 ~ bmi.wave0 + age + sex + dose, data=.) %>%
  summary() %>% coef()
```

Running statistical tests

```
dat.wave0 <- visits %>% filter(wave=="wave0") %>% select(participant, bmi)
dat.wave2 <- visits %>% filter(wave=="wave2") %>% select(participant, bmi)
dat.wave0 %>%
  left_join(dat.wave2, by="participant", suffix=c(".wave0", ".wave2")) %>%
  left_join(participants, by="participant") %>%
  left_join(arms, by="arm") %>%
  lm(bmi.wave2 ~ bmi.wave0 + age + sex + dose, data=.) %>%
  summary() %>% coef()
```

```
              Estimate Std. Error    t value      Pr(>|t|)
(Intercept) -2.47273674 1.77929008 -1.3897322  1.653930e-01
bmi.wave0    1.01448861 0.01752561 57.8860586 4.790746e-195
age          0.08544156 0.06725427  1.2704259  2.046808e-01
sexM         0.08955528 0.14346982  0.6242099  5.328501e-01
dose         0.57515357 0.04672540 12.3092276  1.072305e-29
```

Running statistical tests

```
dat.wave0 <- visits %>% filter(wave=="wave0") %>% select(participant, bmi)
dat.wave2 <- visits %>% filter(wave=="wave2") %>% select(participant, bmi)
dat.wave0 %>%
  left_join(dat.wave2, by="participant", suffix=c(".wave0", ".wave2")) %>%
  left_join(participants, by="participant") %>%
  left_join(arms, by="arm") %>%
  lm(bmi.wave2 ~ bmi.wave0 + age + sex + dose, data=.) %>%
  summary() %>% coef()
```

```
               Estimate Std. Error    t value      Pr(>|t|)
(Intercept) -2.47273674 1.77929008 -1.3897322  1.653930e-01
bmi.wave0    1.01448861 0.01752561 57.8860586 4.790746e-195
age          0.08544156 0.06725427  1.2704259  2.046808e-01
sexM         0.08955528 0.14346982  0.6242099  5.328501e-01
dose         0.57515357 0.04672540 12.3092276  1.072305e-29
```

Treatment dose is strongly associated with increasing BMI (p=1.1e-29).

As expected, BMI is strongly associated across waves (p = 4.8e-195).

Beyond that, age and sex have almost no effect (p > 0.2).

Data pivots

The data transformation we just applied is called a 'pivot from long to wide'.

Data pivots

The data transformation we just applied is called a 'pivot from long to wide'.

1. Initially, the data is **'long'** with one row per BMI measurement per participant.

```
> visits %>% subset(c("participant","wave","bmi"))

   participant wave    bmi
 1           1 wave0  15.4
 2           1 wave1  16.5
 3           1 wave2  14.9
 ...
```

Data pivots

The data transformation we just applied is called a 'pivot from long to wide'.

1. Initially, the data is **'long'** with one row per BMI measurement per participant.

```
> visits %>% subset(c("participant","wave","bmi"))

  participant wave    bmi
1           1 wave0  15.4
2           1 wave1  16.5
3           1 wave2  14.9
...
```

2. After the pivot, the data is **'shorter'** with one row per participant, but **'wider'** due to having two columns providing BMI measurements.

```
> dat.wave0 %>%
  left_join(dat.wave2, by="participant", suffix=c(".wave0", ".wave2"))

  participant bmi.wave0 bmi.wave2
1           1      15.4      14.9
2           2      19.4      21.8
3           3      22.0      23.1
```

Because pivots are so common, tidyr has created `pivot_wider()` and its inverse `pivot_longer()`.

Because pivots are so common, tidyr has created `pivot_wider()` and its inverse `pivot_longer()`.

As before ...

```
## 1. Create a wave 0 subset of 'visits
> dat.wave0 <- visits %>%
  filter(wave=="wave0") %>%
  select(participant, bmi)

## 2. Create a wave 2 subset of 'visits
> dat.wave2 <- visits %>%
  filter(wave=="wave2") %>%
  select(participant, bmi)

## 3. Merge these two
> dat.wave0 %>%
  left_join(dat.wave2,
            by="participant",
            suffix=c(".wave0", ".wave2"))


## 4. Merge this with 'participants' an
  left_join(participants, by="participa
  left_join(arms, by="arm") %>%

## 5. Fit the linear model
  lm(bmi.wave2 ~ bmi.wave0+age+sex+dose
  summary() %>%
  coef()
```

With `pivot_wider` ...

```
## 1-2. Create a wave 0 and 2 subset of
> visits %>%
  filter(wave %in% c("wave0","wave2"))




## 3. pivot dataset (and rename columns
  pivot_wider(
    id_cols=participant,          ## in
    names_from=wave,              ## ne
    values_from=bmi) %>%          ## ne
  rename(bmi.wave0=wave0,bmi.wave2=wave

## 4. Merge this with 'participants' an
  left_join(participants, by="participa
  left_join(arms, by="arm") %>%

## 5. Fit the linear model
  lm(bmi.wave2 ~ bmi.wave0+age+sex+dose
  summary() %>%
  coef()
```

Visualizing data with `ggplot2`

We've looked at a lot of tables of statistics.

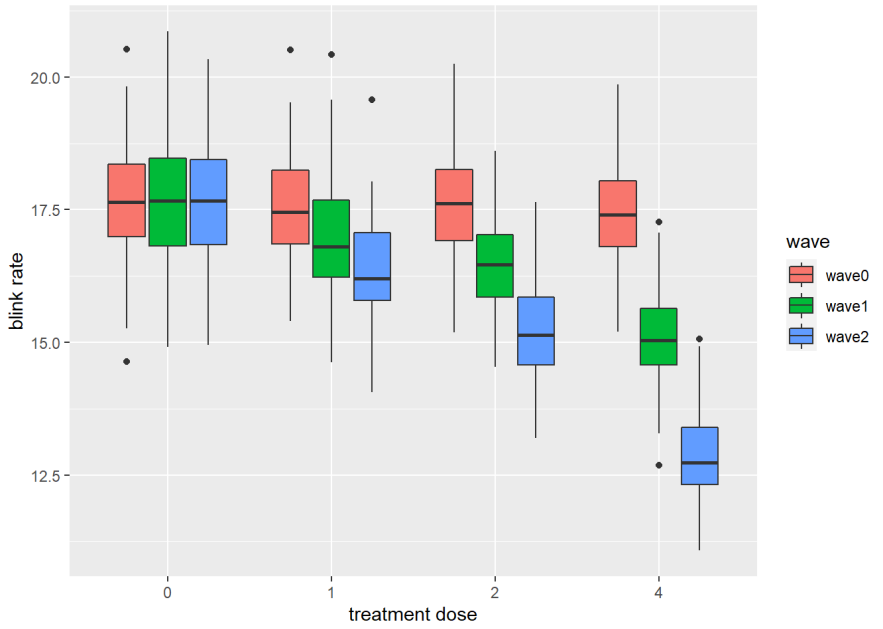
We can visualise these using using `ggplot()`.

Visualizing data with `ggplot2`

We've looked at a lot of tables of statistics.

We can visualise these using using `ggplot()`.

We can visualise blink rate, treatment dose and time as follows:



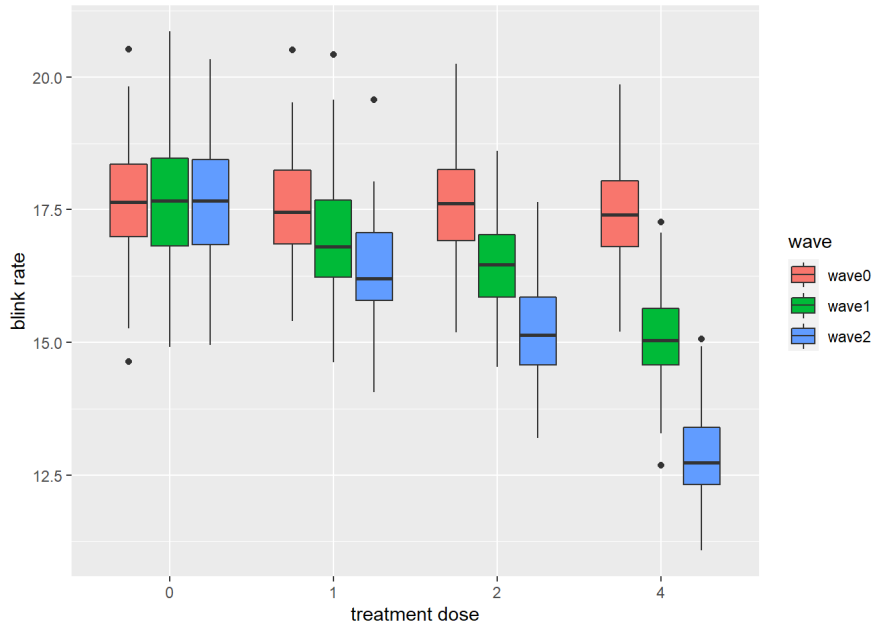
Visualizing data with `ggplot2`

We've looked at a lot of tables of statistics.

We can visualise these using using `ggplot()`.

We can visualise blink rate, treatment dose and time as follows:

1. pipe the data to `ggplot()`



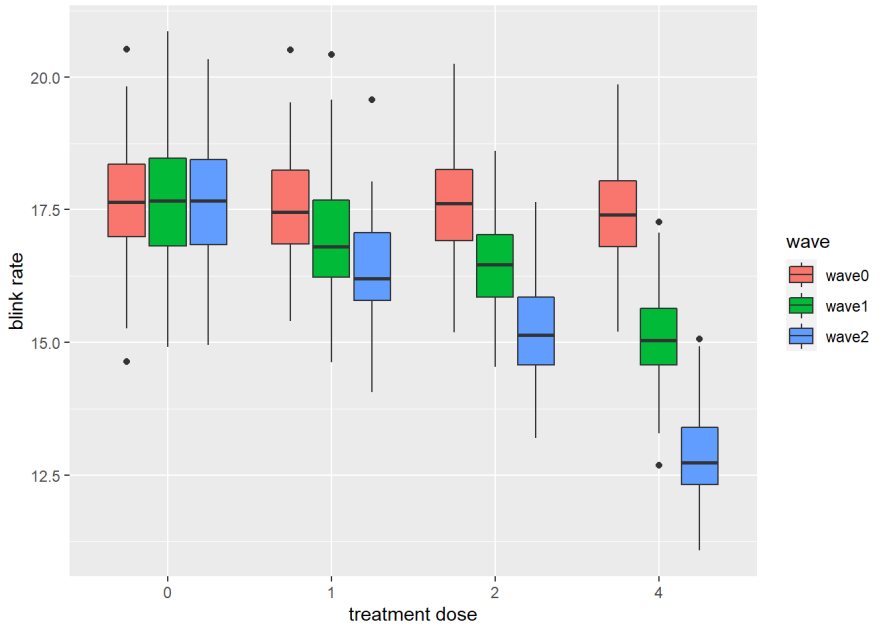
Visualizing data with `ggplot2`

We've looked at a lot of tables of statistics.

We can visualise these using using `ggplot()`.

We can visualise blink rate, treatment dose and time as follows:

1. pipe the data to `ggplot()`
2. tell `ggplot()` to show blink rates (y-a by dose (x-axis) and by wave



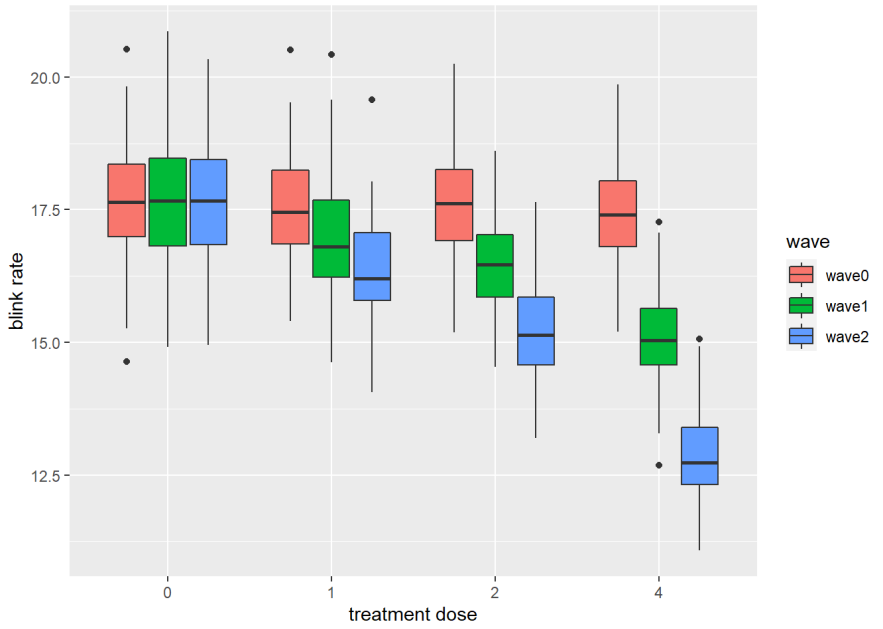
Visualizing data with `ggplot2`

We've looked at a lot of tables of statistics.

We can visualise these using using `ggplot()`.

We can visualise blink rate, treatment dose and time as follows:

1. pipe the data to `ggplot()`
2. tell `ggplot()` to show blink rates (y-a by dose (x-axis) and by wave
3. tell `ggplot` that blink rate distributio displayed as box plots



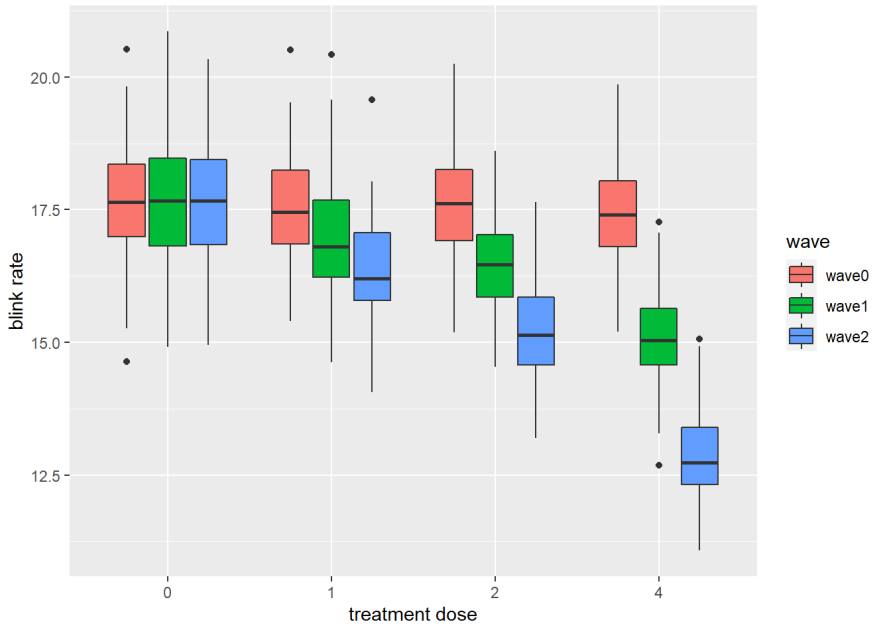
Visualizing data with `ggplot2`

We've looked at a lot of tables of statistics.

We can visualise these using using `ggplot()`.

We can visualise blink rate, treatment dose and time as follows:

1. pipe the data to `ggplot()`
2. tell `ggplot()` to show blink rates (y-a by dose (x-axis) and by wave
3. tell `ggplot` that blink rate distributio displayed as box plots
4. label the x and y axes appropriately



```
## 1. pipe the data to `ggplot()`
visits %>%
  left_join(participants,by="participan
  left_join(arms,by="arm") %>%

## 2. tell `ggplot()` to show blink rat
##    by dose (x-axis) and by wave
  ggplot(aes(x=factor(dose), y=blink, f

## 3. tell `ggplot` that blink rate dis
##    will be displayed as box plots
    geom_boxplot() +

## 4. label the x and y axes appropriat
    xlab("treatment dose") +
    ylab("blink rate")
```

(without the comments)

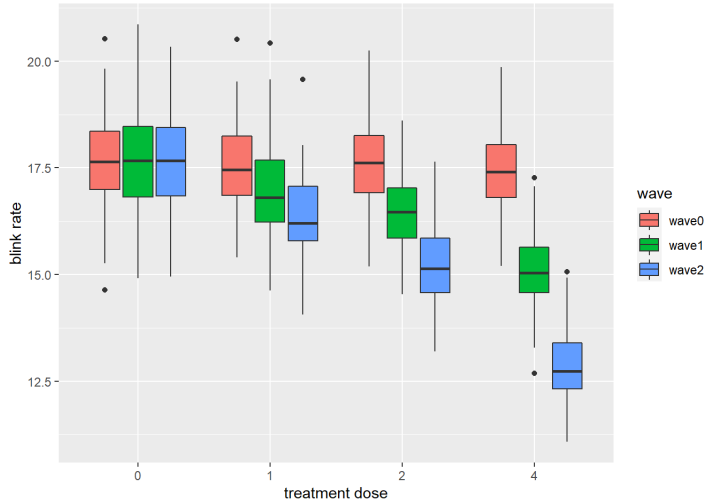
```
visits %>%
  left_join(participants,by="participan
  left_join(arms,by="arm") %>%
  ggplot(aes(x=factor(dose), y=blink, f
    geom_boxplot() +
    xlab("treatment dose") +
    ylab("blink rate")
```

```
## 1. pipe the data to `ggplot()`
visits %>%
  left_join(participants,by="participan
  left_join(arms,by="arm") %>%

## 2. tell `ggplot()` to show blink rat
##    by dose (x-axis) and by wave
  ggplot(aes(x=factor(dose), y=blink, f

## 3. tell `ggplot` that blink rate dis
##    will be displayed as box plots
    geom_boxplot() +

## 4. label the x and y axes appropriat
    xlab("treatment dose") +
    ylab("blink rate")
```



(without the comments)

```
visits %>%
  left_join(participants,by="participan
  left_join(arms,by="arm") %>%
  ggplot(aes(x=factor(dose), y=blink, f
    geom_boxplot() +
    xlab("treatment dose") +
    ylab("blink rate")
```

Talking to ggplot

1. `ggplot` likes "long" data format. If your data is in "wide" format, use `pivot_longer()`.

2. Notice how the '+' is like '%>%'.

    - We pipe *data* to functions with '%>%'.

    - We pipe *instructions* to ggplot with '+'.

    And just like pipes, we can refer to the plot with a variable.

```
p <- visits %>%
  left_join(participants,by="participan
  left_join(arms,by="arm") %>%
  ggplot(aes(x=factor(dose), y=blink, f
    geom_boxplot() +
    xlab("treatment dose") +
    ylab("blink rate")
```

Talking to ggplot

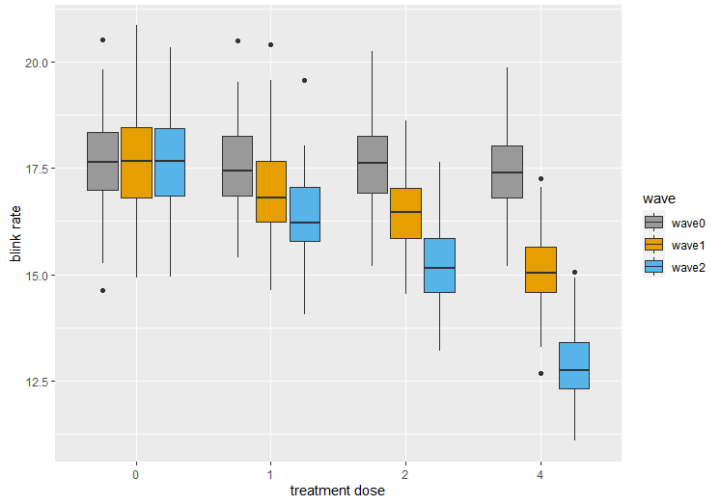
1. ggplot likes "long" data format. If your data is in "wide" format, use pivot_longer().
2. Notice how the '+' is like '%>%'.
   - We pipe *data* to functions with '%>%'.
   - We pipe *instructions* to ggplot with '+'.

   And just like pipes, we can refer to the plot with a variable.

```
p <- visits %>%
  left_join(participants,by="participan
  left_join(arms,by="arm") %>%
  ggplot(aes(x=factor(dose), y=blink, f
    geom_boxplot() +
    xlab("treatment dose") +
    ylab("blink rate")
```

Later, we can add additional instructions about the plot, e.g. to change the boxplot colors

```
fill.cols <- c("#999999", "#E69F00", "#
p <- p + scale_fill_manual(values=fill.
p
```



Challenge!

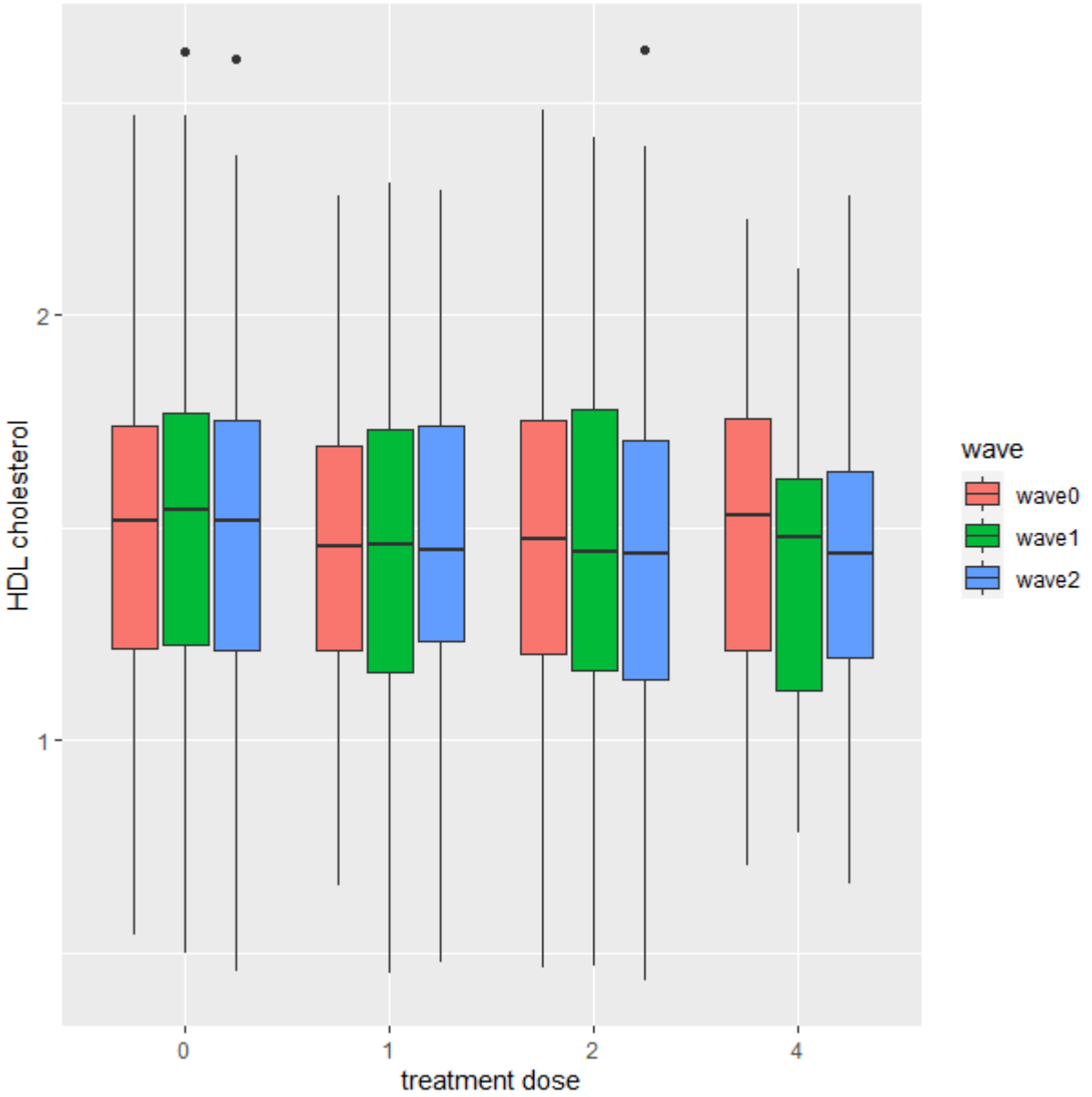
Change the plot to show HDL cholesterol instead of blink rate.

Challenge!

Change the plot to sho
blink rate.

```
visits %>%
  left_join(participa
  left_join(arms,by="
  ggplot(aes(x=factor
   geom_boxplot() +
   xlab("treatment do
   ylab("HDL choleste
```

Sorry! Not very excitin



Challenge!

Generate a plot showing HDL cholesterol vs BMI (baseline).

Hint: use `geom_point()`

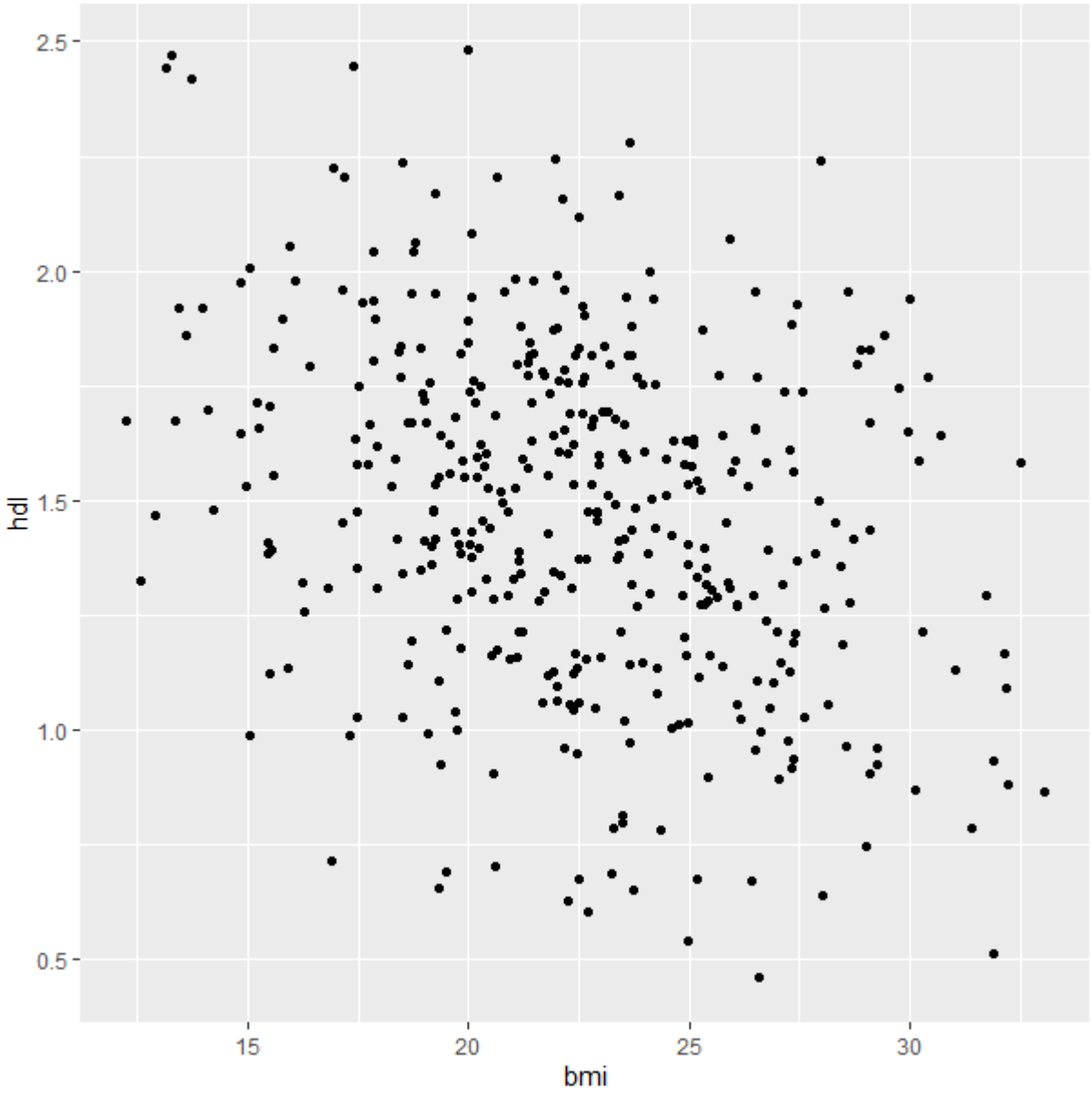
Challenge!

Generate a plot showi

Hint: use `geom_point(`

```
p <- visits %>%
  filter(wave=="wave0
  ggplot(aes(x=bmi, y
   geom_point()
p
```

(saving the plot so we



Challenge!

Generate a plot showi

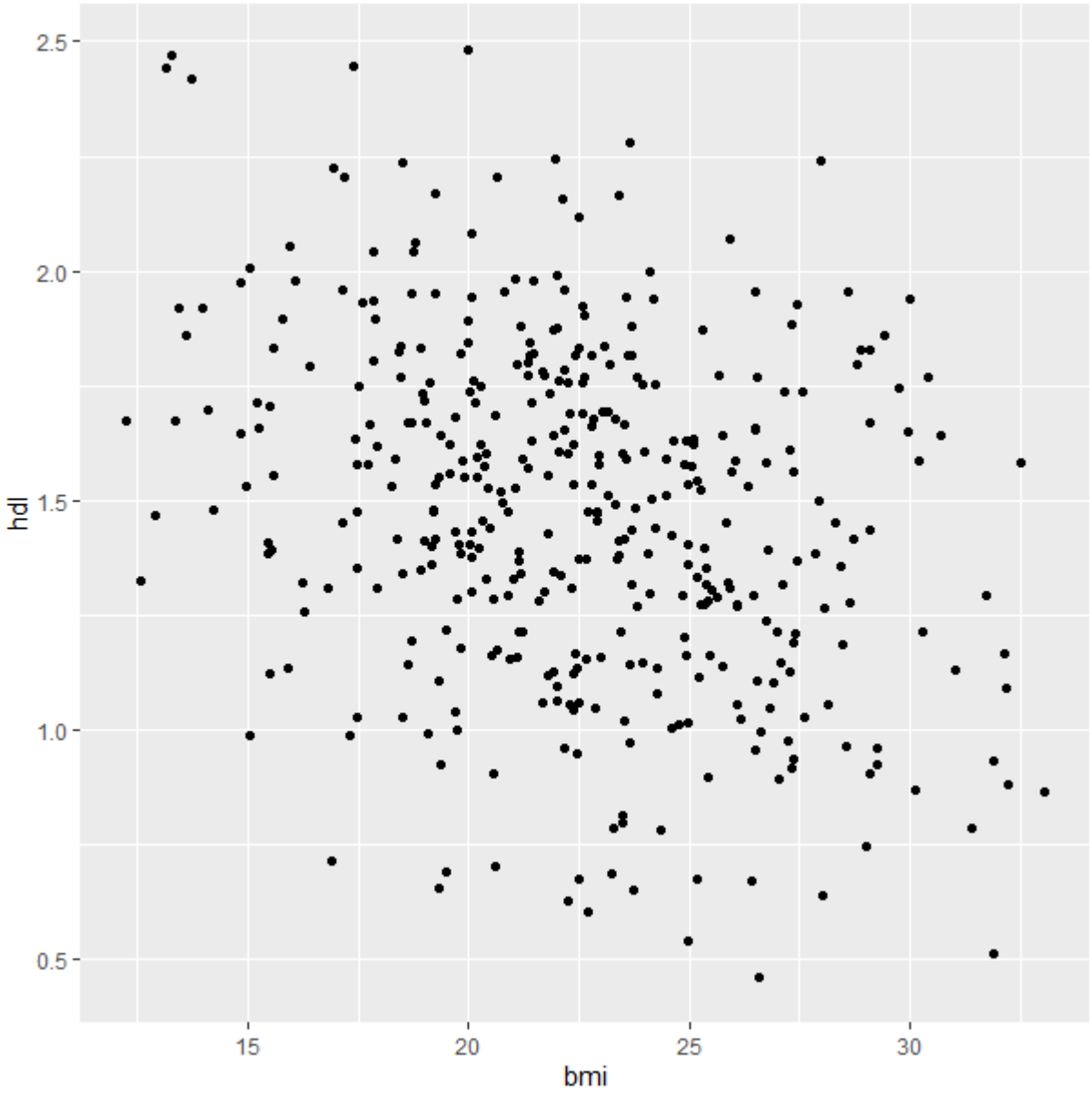
Hint: use `geom_point(`

```
p <- visits %>%
  filter(wave=="wave0
  ggplot(aes(x=bmi, y
   geom_point()
p
```

(saving the plot so we

Now add a regression

Hint: add `geom_smooth`



Challenge!

Generate a plot showi

Hint: use `geom_point(`

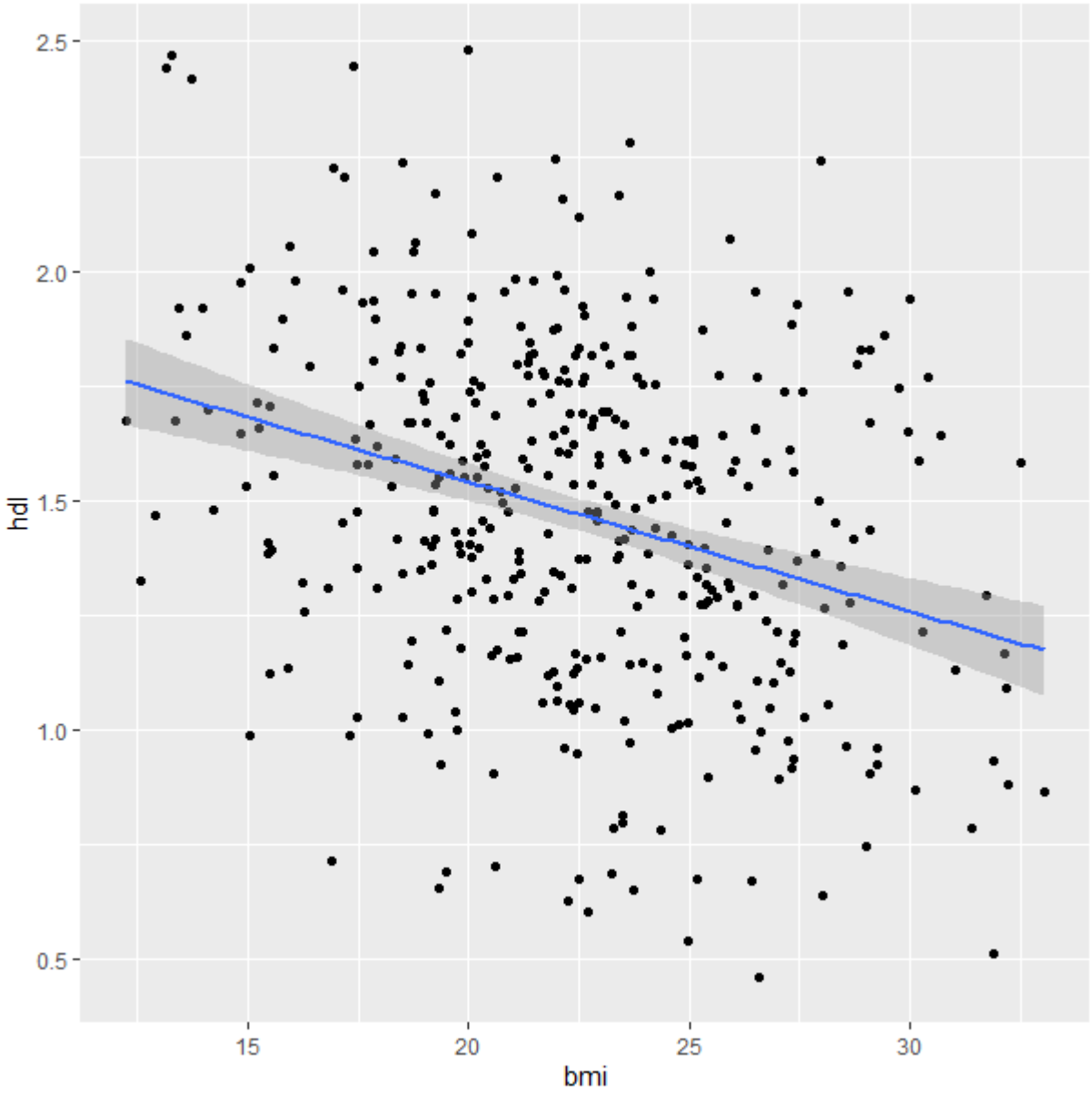
```
p <- visits %>%
  filter(wave=="wave0
  ggplot(aes(x=bmi, y
   geom_point()
p
```

(saving the plot so we

Now add a regression

Hint: add `geom_smooth`

```
p <- p + geom_smooth(
p
```



For more information about the tidyverse

https://r4ds.had.co.nz/

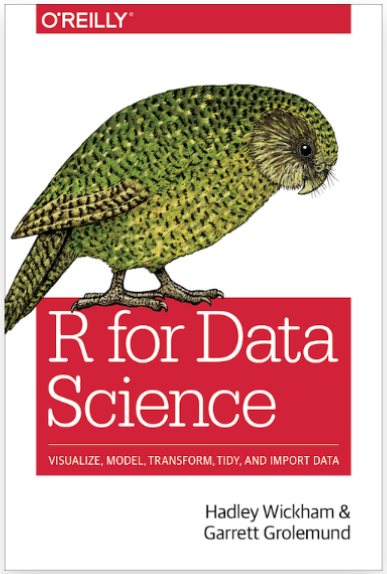
R for Data Science

Search

Table of contents

Welcome

This is the website for **"R for Data Science"**. This book will teach you how to do data science with R: You'll learn how to get your data into R, get it into the most useful structure, transform it, visualise it and model it. In this book, you will find a practicum of skills for data science. Just as a chemist learns how to clean test tubes and stock a lab, you'll learn how to clean data and draw plots—and many other things besides. These are the skills that allow data science to happen, and here you will find the best practices for doing each of these things with R. You'll learn how to use the grammar of graphics, literate programming, and reproducible research to save time. You'll also learn how to manage



Converting between base R and dplyr

If you are familiar with base R, you might find it useful to see how dplyr commands map to base R commands.

https://dplyr.tidyverse.org/articles/base.html

e.g.

dplyr	**base**
arrange(df, x)	df[order(x), , drop = FALSE]
distinct(df, x)	df[!duplicated(x), , drop = FALSE], unique()
filter(df, x)	df[which(x), , drop = FALSE], subset()
mutate(df, z = x + y)	df$z <- df$x + df$y, transform()
pull(df, 1)	df[[1]]
pull(df, x)	df$x
rename(df, y = x)	names(df)[names(df) == "x"] <- "y"
...	

If you choose to enter the tidyverse ...

We've seen how tidyverse attempts to fix some problems in R.

If you choose to enter the tidyverse ...

We've seen how tidyverse attempts to fix some problems in R.

The tidyverse solution isn't without challenges, e.g.

If you choose to enter the tidyverse ...

We've seen how tidyverse attempts to fix some problems in R.

The tidyverse solution isn't without challenges, e.g.

1. tidyverse needs more memorization (eg. dplyr has > 200 functions!)

If you choose to enter the tidyverse ...

We've seen how tidyverse attempts to fix some problems in R.

The tidyverse solution isn't without challenges, e.g.

1. tidyverse needs more memorization (eg. dplyr has > 200 functions!)
2. tidyverse is new and changing (https://dplyr.tidyverse.org/articles/compatibility.html)

If you choose to enter the tidyverse ...

We've seen how tidyverse attempts to fix some problems in R.

The tidyverse solution isn't without challenges, e.g.

1. tidyverse needs more memorization (eg. dplyr has > 200 functions!)
2. tidyverse is new and changing (https://dplyr.tidyverse.org/articles/compatibility.html)
3. tidyverse functions can be difficult to debug

If you choose to enter the tidyverse ...

We've seen how tidyverse attempts to fix some problems in R.

The tidyverse solution isn't without challenges, e.g.

1. tidyverse needs more memorization (eg. dplyr has > 200 functions!)
2. tidyverse is new and changing (https://dplyr.tidyverse.org/articles/compatibility.html)
3. tidyverse functions can be difficult to debug
4. dplyr functions can be much slower than alternatives, e.g. data.table

If you choose to enter the tidyverse ...

We've seen how tidyverse attempts to fix some problems in R.

The tidyverse solution isn't without challenges, e.g.

1. tidyverse needs more memorization (eg. dplyr has > 200 functions!)
2. tidyverse is new and changing
   (https://dplyr.tidyverse.org/articles/compatibility.html)
3. tidyverse functions can be difficult to debug
4. dplyr functions can be much slower than alternatives, e.g. data.table
5. the 'tidyverse way' can be confusing to new R users
   (e.g. tidyverse does a lot 'behind the scenes magic' that can seem mysterious)

https://github.com/matloff/TidyverseSkeptic