PS947: LECTURE 2. TIDYVERSE

A.E. Hughes & A.D.F Clarke

THE PLAN THIS WEEK

Today:

- R and linear models: revision and extension of last week's work
- An introduction to data wrangling

Thursday:

- An introduction to plotting

1. R AND LINEAR MODELS

OVERVIEW

1. Create a folder and file
2. Specify your hypothesis
3. Data simulation
4. Power/sensitivity analysis
5. Collect data
6. Analyse data

MAKING FOLDERS - REVISION

- Open RStudio and create a new empty .R file
- Save it as `week2_examples.R` in `week2/scripts`
- Set the Working Directory to `week2/scripts`
- We are now ready to start!

AT THE START OF EACH FILE - REVISION

```r
# Load "tidyverse"
library(tidyverse)

# Set some options
options(scipen = 5, digits = 3)

# There are new helper functions this week
source("week2_helper.R")
```

AT THE START OF EACH FILE - SET.SEED

Generating random numbers on a computer is a surprisingly difficult problem!

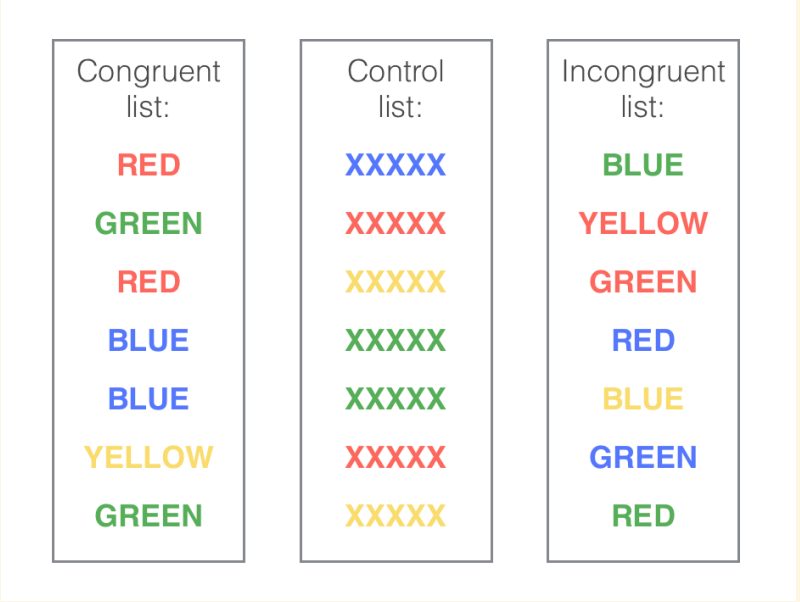
`set.seed` sets the starting number used to generate a sequence of pseudo-random numbers.

Means that you'll get the same results if you start with the same seed (good for reproducibility).

```
1  set.seed(947)
```

RESEARCH QUESTION

Is there a difference in reaction time between congruent and incongruent stimuli?



A probably familiar stimulus

WHAT EFFECT SIZE MIGHT WE EXPECT?

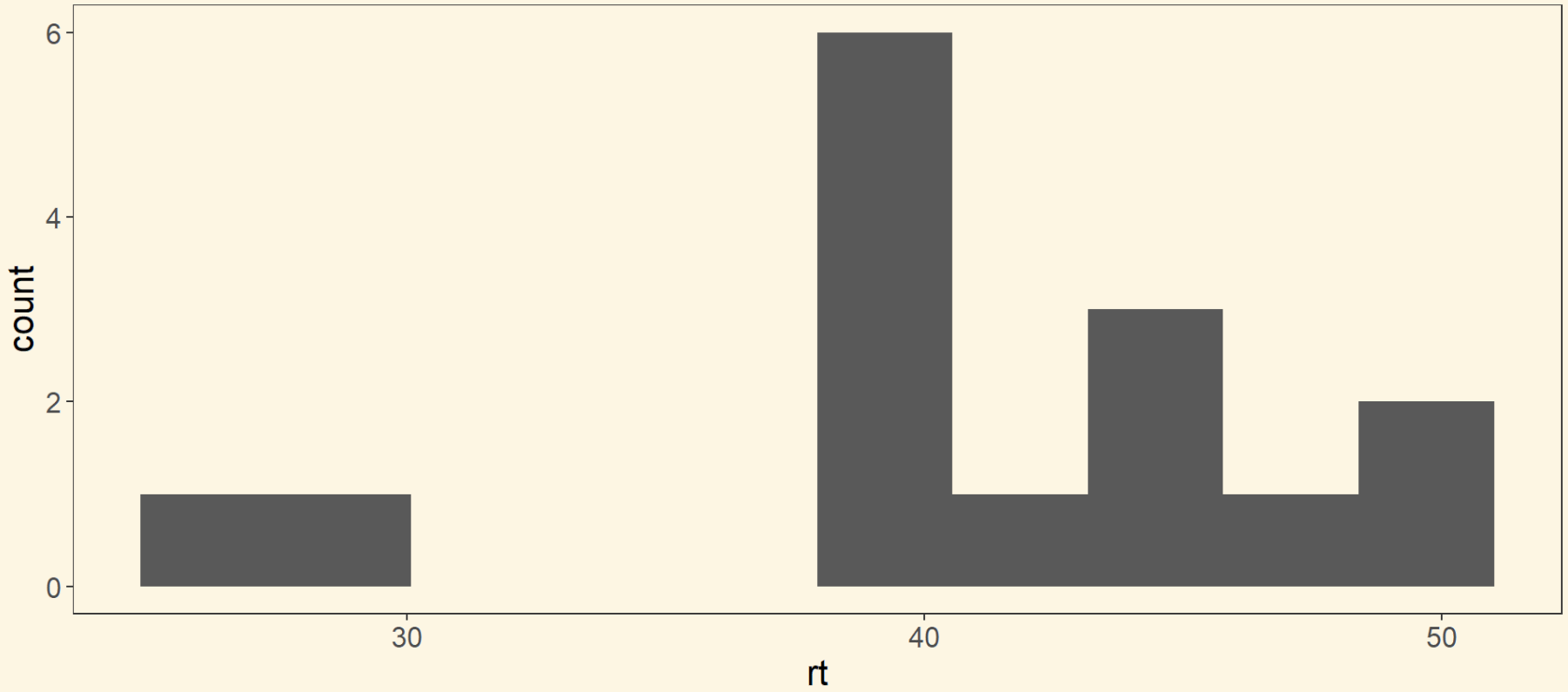
There is a lot of previous Stroop task data! We might want to use it while carrying out our simulations for power analysis.

What might make sensible estimates of the means and standard deviations for the **congruent** and **incongruent** conditions?

- Find a paper that uses the Stroop task and try to find this information.

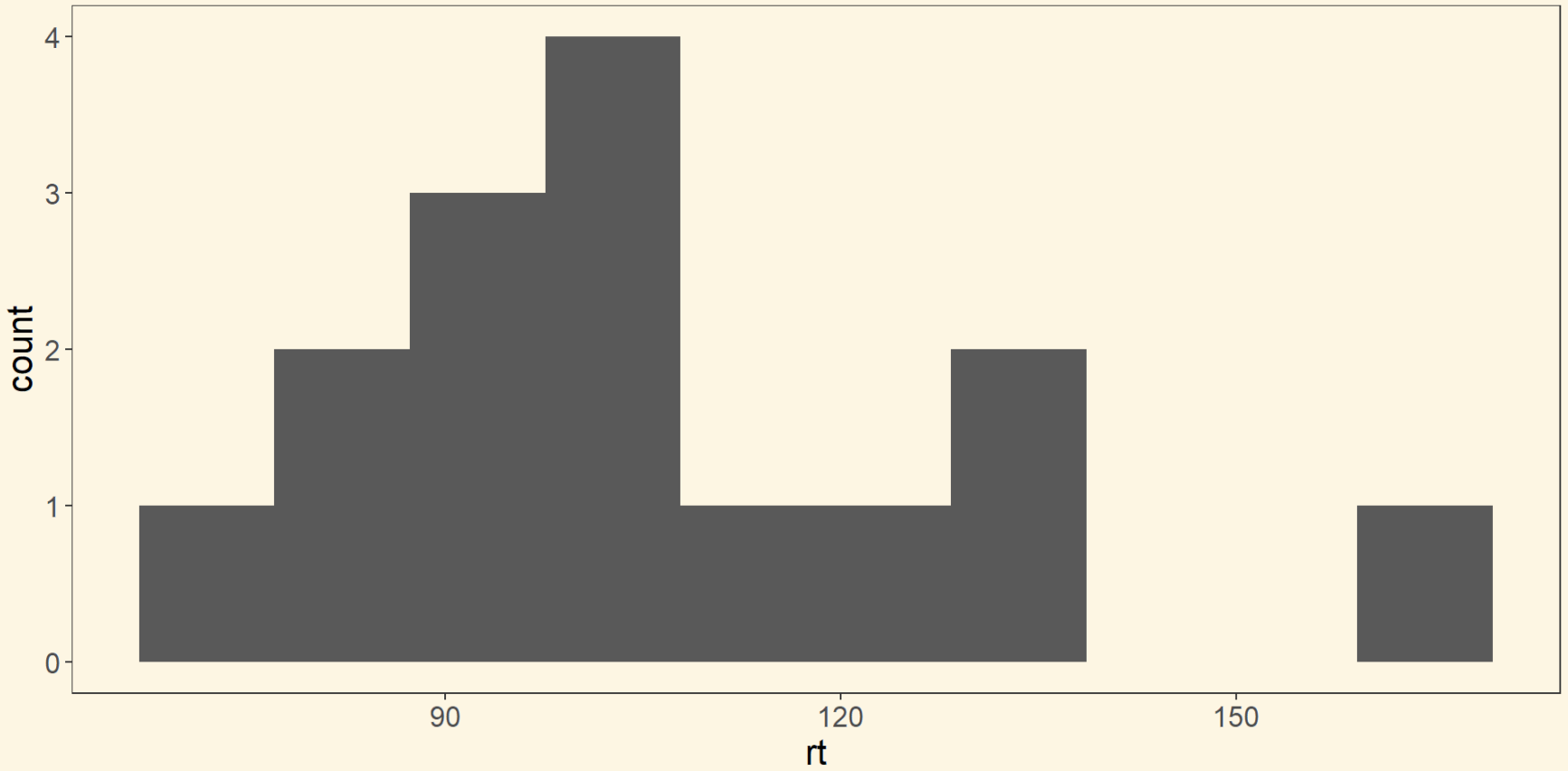
SIMULATING DATA: CREATING OUR CONGRUENT DATA

```r
n <- 15
congruent_data <- tibble(
  rt = rnorm(n, mean = 43.3, sd = 6.15))

ggplot(congruent_data, aes(rt)) +
  geom_histogram(bins = 10)
```



SIMULATING DATA: CREATING OUR INCONGRUENT DATA

```r
incongruent_data <- tibble(
  rt = rnorm(n, mean = 110.3, sd = 18.8))

ggplot(incongruent_data, aes(rt)) +
  geom_histogram(bins = 10)
```



MAKING A TIBBLE OF OUR DATA

- The $ allows us to pull out a specific column from the tibble
- We can make new variables!

```
data <- tibble(
  congruent = congruent_data$rt,
  incongruent = incongruent_data$rt,
  diff = incongruent - congruent)
```

USING A LINEAR MODEL FOR ANALYSIS

Remember our linear model: $y = mx + c$

We have the pairwise differences and we want to test whether they are different from zero on average.

In terms of a linear model, this simply means we test whether the **intercept** is different from zero. We have no x variable, so our model simplifies to $y = c$.

POWER ANALYSIS

```r
model <- lm(diff ~ 1, data = data)
summary(model)
```

```
Call:
lm(formula = diff ~ 1, data = data)

Residuals:
   Min     1Q Median     3Q    Max
-36.37 -12.51  -7.03   6.27  52.80

Coefficients:
            Estimate Std. Error t value    Pr(>|t|)
(Intercept)    65.29       6.22    10.5 0.000000051 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 24.1 on 14 degrees of freedom
```

POWER ANALYSIS - ITERATE

This is similar but not identical to last week!

```r
simple_power_analysis_l2 <- function(n, mean, sd) {

  p_values = array()

  for (itr in 1:100){

    # Slides 9-11
    d <- tibble(diff = rnorm(n, mean, sd))

    # Slides 12-13
    m <- lm(diff ~ 1, data = d)

    p_values[itr] <- get_p_value_l2(m)

  }

  return(mean(p_values < 0.05))

}
```

FULL POWER ANALYSIS

Let's start by using the values I estimated with my model on previous data.

```
simple_power_analysis_l2(n = 15, mean = 65.29, sd = 6.22)
```

```
[1] 1
```

- What happens if you change the sample size, mean difference, standard deviation?
- What happens if you use estimates from the papers you found?
- If you were doing a power analysis for a real experiment, what might be the problems with this approach?

LET'S INTRODUCE ANOTHER LOOP...

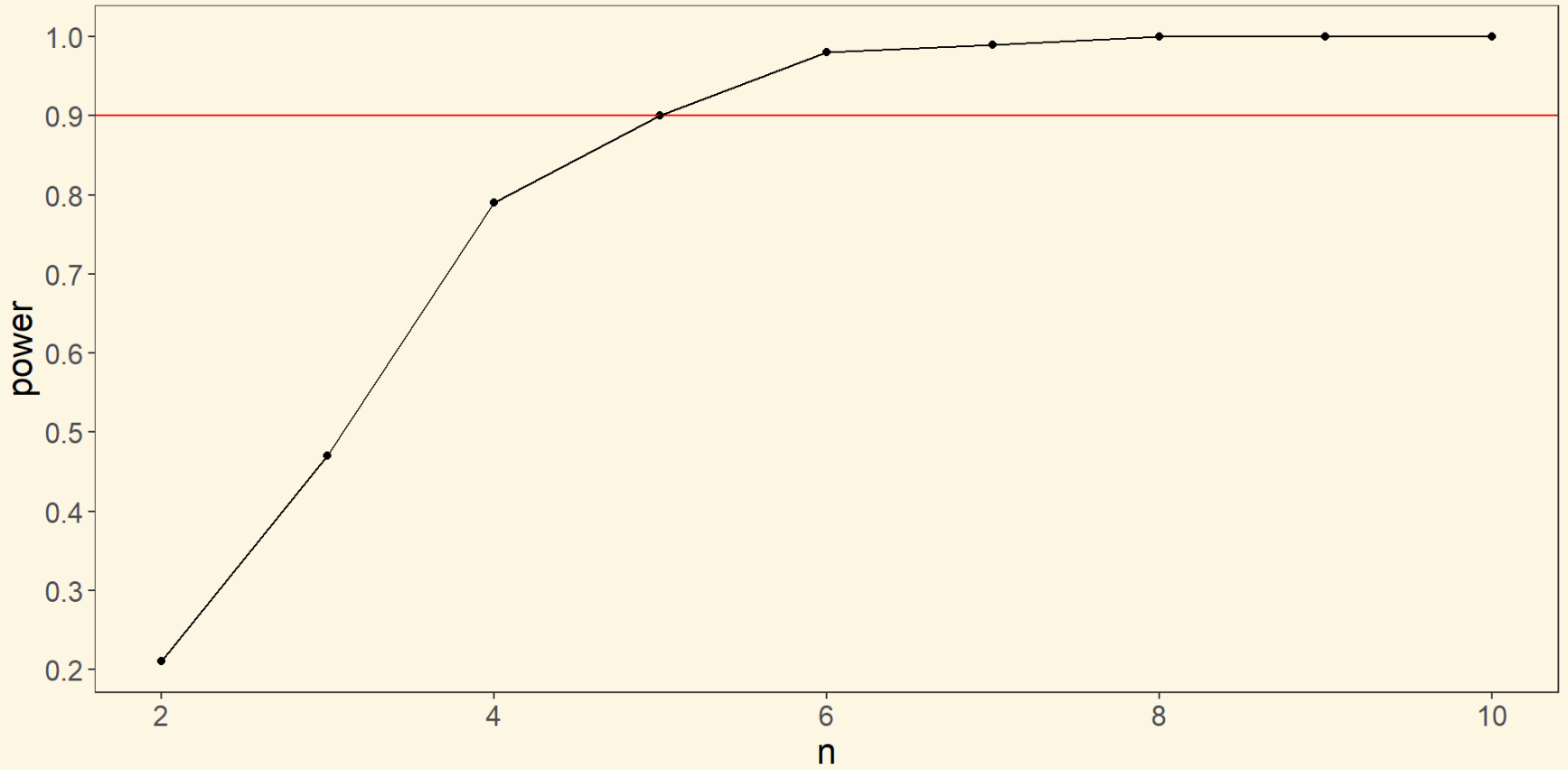
It's a bit boring having to manually enter different sample sizes.

The **map** function allows us to apply our function over a range of different n values.

```
d_power <- tibble(
  n = 2:10,
  power = map_dbl(n, simple_power_analysis_l2,
                  mean = 10, sd = 5))
```

PLOTTING OUR RESULTS

```
ggplot(d_power, aes(n, power)) +
  geom_point() +
  geom_hline(yintercept = 0.9, colour = 'red') +
  geom_path() +
  scale_y_continuous(breaks = seq(0, 1, len = 11))
```



REAL DATA

We need to collect it!

Let’s do it…

SOME STIMULI

RED	GREEN	BLACK	ORANGE
GREEN	PURPLE	RED	GREEN
ORANGE	BLACK	BLUE	RED
BLUE	RED	ORANGE	BLACK
PURPLE	BLUE	GREEN	PURPLE
BLACK	ORANGE	PURPLE	BLUE
GREEN	BLACK	BLUE	GREEN
RED	BLUE	RED	PURPLE
PURPLE	ORANGE	PURPLE	ORANGE
BLUE	GREEN	BLACK	RED
BLACK	RED	GREEN	BLUE
ORANGE	PURPLE	ORANGE	BLACK

ANALYSING THE REAL DATA

- Make a tibble with your real data
- Work out some descriptive statistics
- Fit an appropriate model
- What can you conclude?

BREAK TIME AND A *STRETCH GOAL*

Time for a short break!

A *stretch goal* for you to think about: how would you go about doing a power analysis for your MSc dissertation/first PhD project?

(We are happy to talk about this with you if you'd like to attempt it!)

2. AN INTRODUCTION TO DATA WRANGLING

DATA WRANGLING

Very often, we have raw data that we want to 'tidy up' before we start analysis (data cleaning, preprocessing etc.)

The `tidyverse` package contains some useful functions for helping us to carry this out.

We'll go through some of these functions now, and practice using them.

DATA WRANGLING: LOADING A DATASET

The `starwars` dataset is included as part of the `tidyverse` packages.

```
1 starwars
```

```
# A tibble: 87 × 14
   name      height  mass hair_color skin_color eye_color birth_year sex
gender
   <chr>      <int> <dbl> <chr>      <chr>      <chr>           <dbl> <chr>
<chr>
 1 Luke Sk…    172    77 blond      fair       blue             19   male
mascu…
 2 C-3PO       167    75 <NA>       gold       yellow          112   none
mascu…
 3 R2-D2        96    32 <NA>       white, bl… red              33   none
mascu…
 4 Darth V…    202   136 none       white      yellow           41.9 male
mascu…
 5 Leia Or…    150    49 brown      light      brown            19   fema…
femin…
```

FILTERING

Subset observations based on their values.

```
filter(starwars, hair_color == "brown")
```

```
# A tibble: 18 × 14
   name      height  mass hair_color skin_color eye_color birth_year sex
gender
   <chr>      <int> <dbl> <chr>      <chr>      <chr>           <dbl> <chr>
<chr>
 1 Leia Or…    150    49 brown      light      brown              19 fema…
femin…
 2 Beru Wh…    165    75 brown      light      blue               47 fema…
femin…
 3 Chewbac…    228   112 brown      unknown    blue              200 male
mascu…
 4 Han Solo    180    80 brown      fair       brown              29 male
mascu…
 5 Wedge A…    170    77 brown      fair       hazel              21 male
mascu…
```

Note the double equals! This means “is equal to”. (Check workbook 1 if you don’t remember).

FILTERING MISSING VALUES

In R, missing values should be denoted by NA. Sometimes, you'll want to filter out rows where there are missing values, and there's a helpful function to do this quickly.

```
1 # We can filter out any row that has a missing value in any column...
2 drop_na(starwars)
```

```
# A tibble: 29 × 14
   name     height  mass hair_color skin_color eye_color birth_year sex
gender
   <chr>     <int> <dbl> <chr>      <chr>      <chr>           <dbl> <chr>
<chr>
 1 Luke Sk…    172    77 blond      fair       blue             19   male
mascu…
 2 Darth V…    202   136 none       white      yellow           41.9 male
mascu…
 3 Leia Or…    150    49 brown      light      brown            19   fema…
femin…
 4 Owen La…    178   120 brown, gr… light      blue             52   male
mascu…
 5 Beru Wh…    165    75 brown      light      blue             47   fema…
femin…
```

FILTERING MISSING VALUES

In R, missing values should be denoted by NA. Sometimes, you'll want to filter out rows where there are missing values, and there's a helpful function to do this quickly.

```
# ...Or specify a specific column
drop_na(starwars, hair_color)
```

```
# A tibble: 82 × 14
   name      height  mass hair_color skin_color eye_color birth_year sex
gender
   <chr>      <int> <dbl> <chr>      <chr>      <chr>           <dbl> <chr>
<chr>
 1 Luke Sk…    172    77 blond      fair       blue             19   male
mascu…
 2 Darth V…    202   136 none       white      yellow           41.9 male
mascu…
 3 Leia Or…    150    49 brown      light      brown            19   fema…
femin…
 4 Owen La…    178   120 brown, gr… light      blue             52   male
mascu…
 5 Beru Wh…    165    75 brown      light      blue             47   fema…
femin…
```

FILTERING - YOUR TURN

What do the following do?

```
1 filter(starwars, height > 150)
2
3 filter(starwars, species == "Human", skin_color != "fair")
4
5 filter(starwars, is.na(homeworld))
```

SELECTING

Here's how to select a particular column. Particularly useful if your dataset has huge numbers of variables!

```
 1 select(starwars, name, films, homeworld)
```

```
# A tibble: 87 × 3
   name               films     homeworld
   <chr>              <list>    <chr>
 1 Luke Skywalker     <chr [5]> Tatooine
 2 C-3PO              <chr [6]> Tatooine
 3 R2-D2              <chr [7]> Naboo
 4 Darth Vader        <chr [4]> Tatooine
 5 Leia Organa        <chr [5]> Alderaan
 6 Owen Lars          <chr [3]> Tatooine
 7 Beru Whitesun lars <chr [3]> Tatooine
 8 R5-D4              <chr [1]> Tatooine
 9 Biggs Darklighter  <chr [1]> Tatooine
10 Obi-Wan Kenobi     <chr [6]> Stewjon
# ℹ 77 more rows
```

UNNESTING

You may have noticed that the `film` column looks a bit odd - this is because each cell contains a list. If we want to see each entry of that list, we can use the `unnest` function.

```
1 starwars_short <- select(starwars, name, films, homeworld)
2 unnest(starwars_short, cols = films)
```

```
# A tibble: 173 × 3
   name           films                     homeworld
   <chr>          <chr>                     <chr>
 1 Luke Skywalker The Empire Strikes Back   Tatooine
 2 Luke Skywalker Revenge of the Sith       Tatooine
 3 Luke Skywalker Return of the Jedi        Tatooine
 4 Luke Skywalker A New Hope                Tatooine
 5 Luke Skywalker The Force Awakens         Tatooine
 6 C-3PO          The Empire Strikes Back   Tatooine
 7 C-3PO          Attack of the Clones      Tatooine
 8 C-3PO          The Phantom Menace        Tatooine
 9 C-3PO          Revenge of the Sith       Tatooine
10 C-3PO          Return of the Jedi        Tatooine
# ℹ 163 more rows
```

SELECTING - YOUR TURN

What do the following do?

```
1 select(starwars, name:birth_year)
2
3 select(starwars, ends_with("_color"))
```

MUTATING

Mutate allows you to add new variables.

```
1 mutate(starwars, norm_mass = mass/height)
```

```
# A tibble: 87 × 15
   name      height  mass hair_color skin_color eye_color birth_year sex
gender
   <chr>      <int> <dbl> <chr>      <chr>      <chr>           <dbl> <chr>
<chr>
 1 Luke Sk…    172    77 blond      fair       blue             19   male
mascu…
 2 C-3PO       167    75 <NA>       gold       yellow          112   none
mascu…
 3 R2-D2        96    32 <NA>       white, bl… red              33   none
mascu…
 4 Darth V…    202   136 none       white      yellow          41.9 male
mascu…
 5 Leia Or…    150    49 brown      light      brown            19   fema…
femin…
 6 Owen La…    178   120 brown, gr… light      blue             52   male
```

MUTATING

- New columns are added to the end of the dataset
- You can refer to columns you've just created

```
1 mutate(starwars, norm_mass = mass/height, silly = norm_mass + 1)
```

```
# A tibble: 87 × 16
   name     height  mass hair_color skin_color eye_color birth_year sex
gender
   <chr>     <int> <dbl> <chr>      <chr>      <chr>           <dbl> <chr>
<chr>
 1 Luke Sk…    172    77 blond      fair       blue             19   male
mascu…
 2 C-3PO       167    75 <NA>       gold       yellow          112   none
mascu…
 3 R2-D2        96    32 <NA>       white, bl… red              33   none
mascu…
 4 Darth V…    202   136 none       white      yellow           41.9 male
mascu…
 5 Leia Or…    150    49 brown      light      brown            19   fema…
femin…
```

MUTATION OPERATIONS

SUMMARISING

Summarising your data is (generally) a two-step process:

- use **group_by** to group your dataset by specific variables
- apply **summarise** to summarise by these variables

```
grouping <- group_by(starwars, gender)
summarise(grouping, mean_mass = mean(mass, na.rm = TRUE), n = n())
```

```
# A tibble: 3 × 3
  gender    mean_mass     n
  <chr>         <dbl> <int>
1 feminine       54.7    17
2 masculine     106.     66
3 <NA>           48       4
```

SUMMARISING

```r
grouping2 <- group_by(starwars, eye_color, gender)
summarise(grouping2, mean_mass = mean(mass, na.rm = TRUE), n = n())
```

```
# A tibble: 23 × 4
# Groups:   eye_color [15]
   eye_color gender    mean_mass     n
   <chr>     <chr>         <dbl> <int>
 1 black     feminine       57       2
 2 black     masculine      79.5     8
 3 blue      feminine       57.8     6
 4 blue      masculine     101.     12
 5 blue      <NA>          NaN       1
 6 blue-gray masculine      77       1
 7 brown     feminine       47       5
 8 brown     masculine      69.6    15
 9 brown     <NA>          NaN       1
10 dark      masculine     NaN       1
# ℹ 13 more rows
```

USEFUL SUMMARY FUNCTIONS

- **Measures of location:** mean(x), median(x)
- **Measures of spread:** sd(x), IQR(x), mad(x) (a robust equivalent - useful if you have outliers!)
- **Measures of rank:** min(x), quantile(x, 0.25), max(x)
- **Measures of position:** first(x), nth(x,2), last(x)
- **Counts:** n(), n_distinct()
- **Counts and proportions of logical values:** sum(x) gives number of TRUEs in x, mean(x) gives proportion

COMBINING MULTIPLE OPERATIONS: THE PIPE

In many cases, you want to carry out multiple operations on your dataset e.g. group, summarise and then mutate.

The pipe can help you do this!

It can be thought of as a "**then**" command.

Its syntax is %>%.

A PIPE EXAMPLE

```r
pipe_example <- starwars %>%
  group_by(eye_color, gender) %>%
  summarise(mean_mass = mean(mass, na.rm = TRUE), n = n()) %>%
  mutate(nonsense_variable = mean_mass/n)

head(pipe_example)
```

```
# A tibble: 6 × 5
# Groups:   eye_color [3]
  eye_color gender    mean_mass     n nonsense_variable
  <chr>     <chr>         <dbl> <int>              <dbl>
1 black     feminine       57       2              28.5
2 black     masculine      79.5     8               9.94
3 blue      feminine       57.8     6               9.63
4 blue      masculine     101.     12               8.41
5 blue      <NA>          NaN       1             NaN
6 blue-gray masculine      77       1              77
```

COMBINING MULTIPLE OPERATIONS - YOUR TURN

Have a go at the following:

- Select the `name`, `starships` and `birth_year` columns in the `starwars` data
- Unnest the `starships` column
- Remove rows with NAs in the `birth_year` column

Can you do it without the pipe (`%>%`)? With the pipe? What are the advantages and disadvantages of these different approaches?

COMBINING MULTIPLE OPERATIONS - YOUR TURN

Have a go at the following:

- Look at the msleep dataset from package `ggplot2` (this is also a `tidyverse` package, so should already be loaded!) Type `?msleep` in the console to find out more about what the data means.
- Filter by conservation status `vu`
- Group by `vore`
- Work out the counts in each group, and the mean sleep total, and the mean body weight

HOMEWORK

- Remember your first homework is due at 1pm on **Fri 2nd Feburary**
- You can find it, and the data you need to complete it, on Moodle
- It covers material from weeks 1 and 2
- We are happy to offer general support - ask us on Discord
- Remember that material labelled as ‘distinction material’ is meant to be hard! If you get stuck or run out of time, focus on the other questions - you will still be able to get a good mark by completing just those parts well

3. AN INTRODUCTION TO PLOTTING

PLOTTING IN R - GGPLOT2

All graphs in ggplot2 follow the same template:

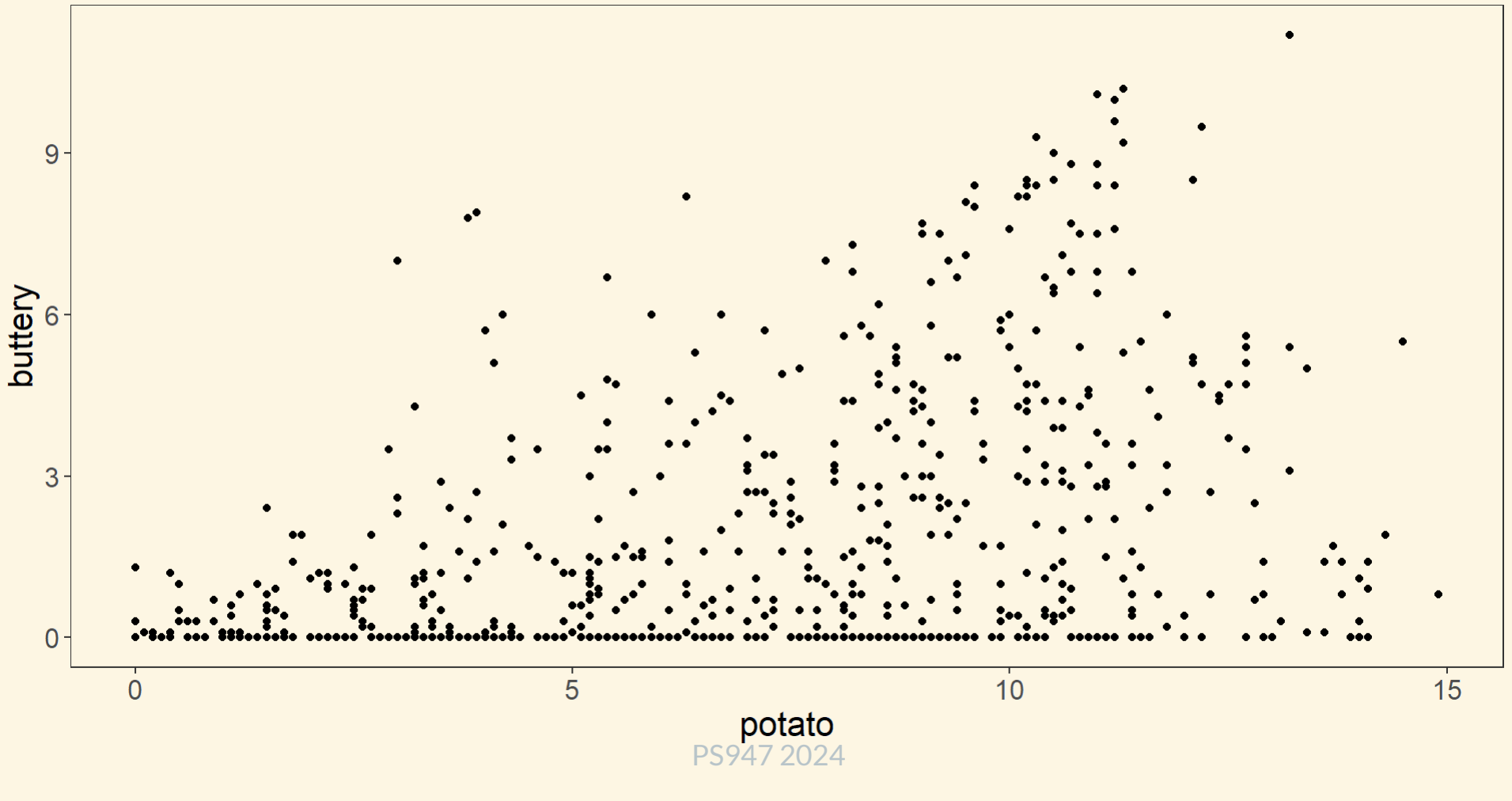
ggplot(data = **DATA**) + **GEOM_FUNCTION**(mapping = aes(**MAPPINGS**))

GEOM_FUNCTION: adds one or more layers e.g. *type* of graph (scatterplot, bar graph, boxplot etc.)

MAPPINGS: define how variables in your dataset are mapped to visual properties

A GGPLOT EXAMPLE

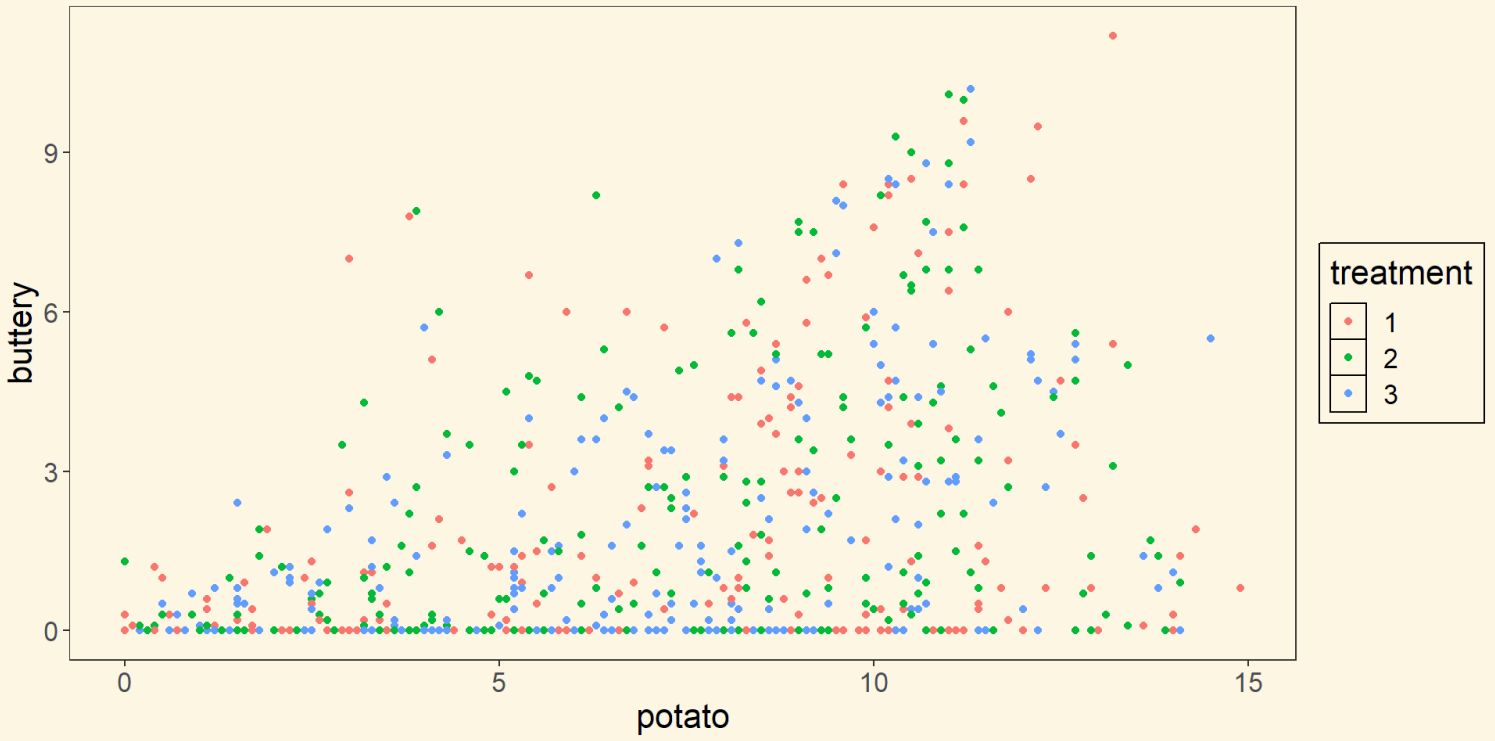
```
1  library(reshape)
2
3  ggplot(data = french_fries) +
4    geom_point(mapping = aes(x = potato, y = buttery))
```



AESTHETIC MAPPINGS

Aesthetics: visual properties of objects in your plot e.g. size, shape, colour

```
ggplot(data = french_fries) +
  geom_point(mapping = aes(x = potato, y = buttery,
                           colour = treatment))
```



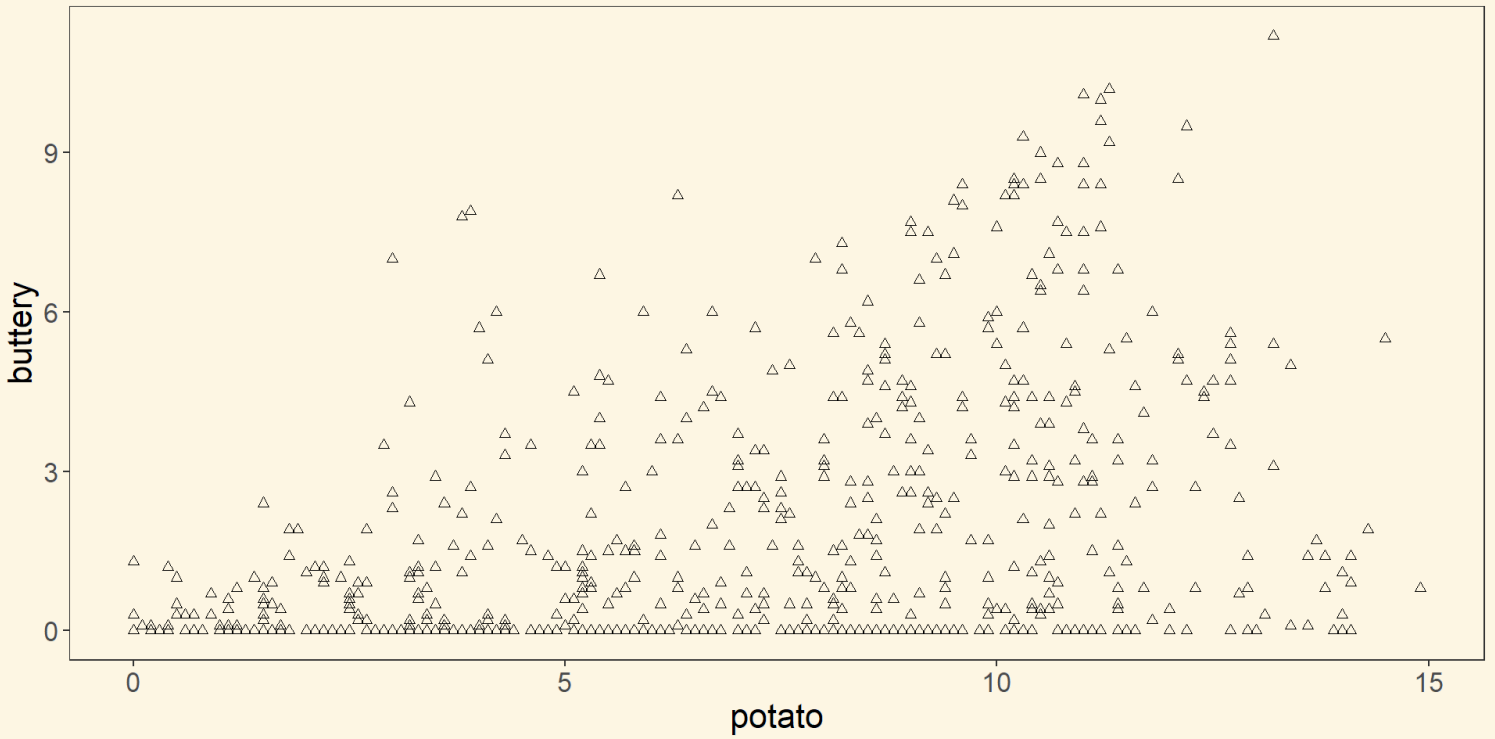
SETTING AESTHETIC MAPPINGS MANUALLY

If you set your aesthetics within the aes() brackets, ggplot does lots of things automatically - selects appropriate scales, creates a legend etc.

SETTING AESTHETIC MAPPINGS MANUALLY

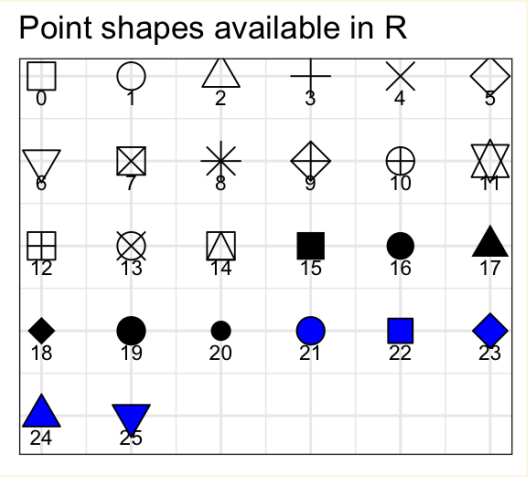
But you can also set aesthetics manually: in this case, set them outside of the aes() brackets.

```
ggplot(data = french_fries) +
  geom_point(mapping = aes(x = potato, y = buttery),
             shape = 2)
```



YOUR TURN

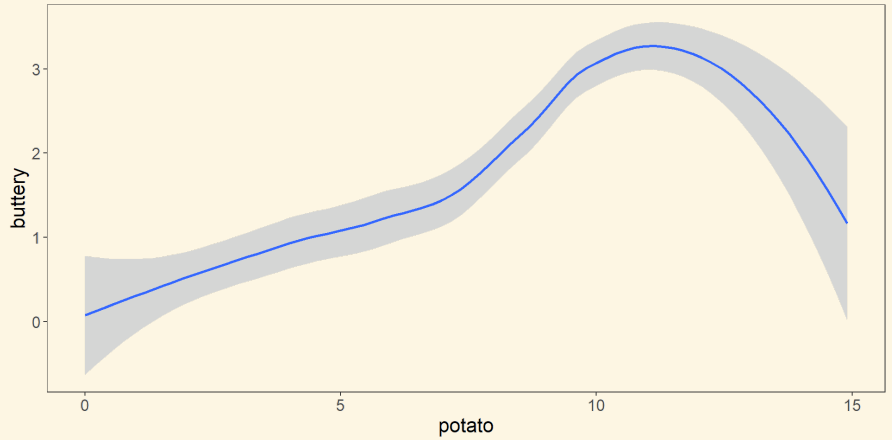
- Set all the points of the graph to have the colour blue
- Try out some different numbers for shape (see possibilities below)
- Try setting `alpha = 0.5` outside of the aes() brackets. What does this do?



GEOMETRIC OBJECTS - GEOMS

Geoms can be thought of as the “type” of graph you are drawing. You can use different geoms to plot the same data.

```
1 ggplot(french_fries) +
2   geom_smooth(aes(potato, buttery))
```



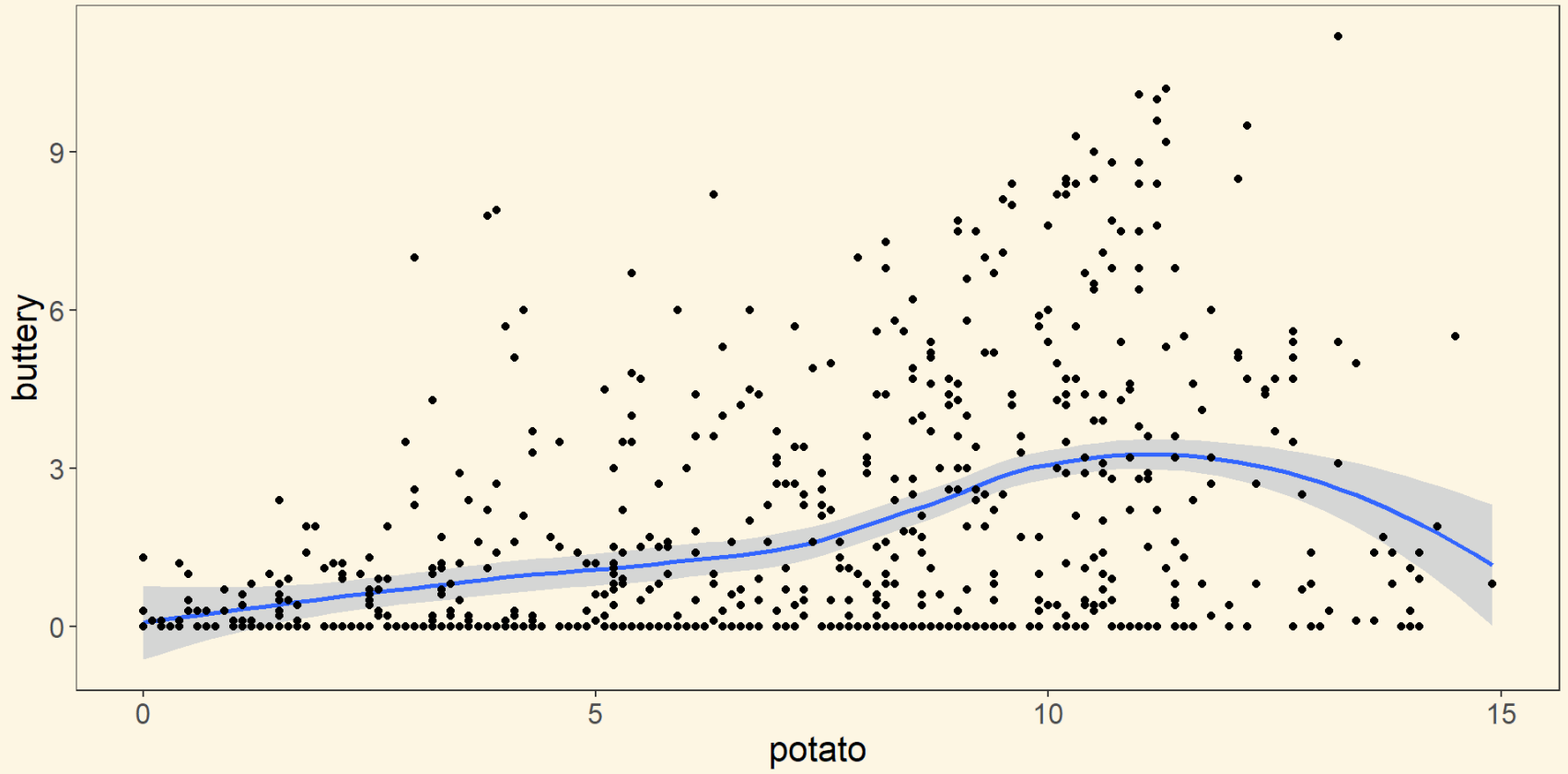
Note: not all aesthetics work with all geoms!

Meaningless to talk about “shape” of a line - but you can set “linetype”.

DISPLAYING MULTIPLE GEOMS IN A PLOT

The way ggplot works makes it very easy to "layer" geoms.

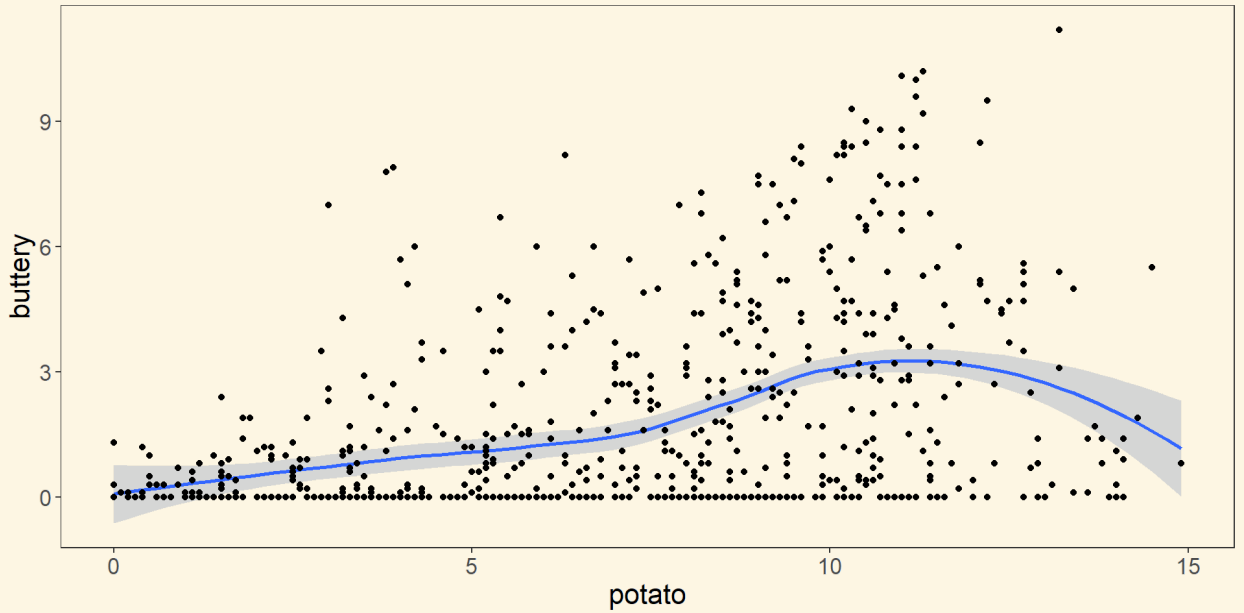
```
1  ggplot(french_fries) +
2    geom_smooth(aes(potato, buttery)) +
3    geom_point(aes(potato, buttery))
```



DISPLAYING MULTIPLE GEOMS IN A PLOT

The way ggplot works makes it very easy to "layer" geoms.

```
1  ggplot(french_fries, aes(potato, buttery)) +
2    geom_smooth() +
3    geom_point()
```

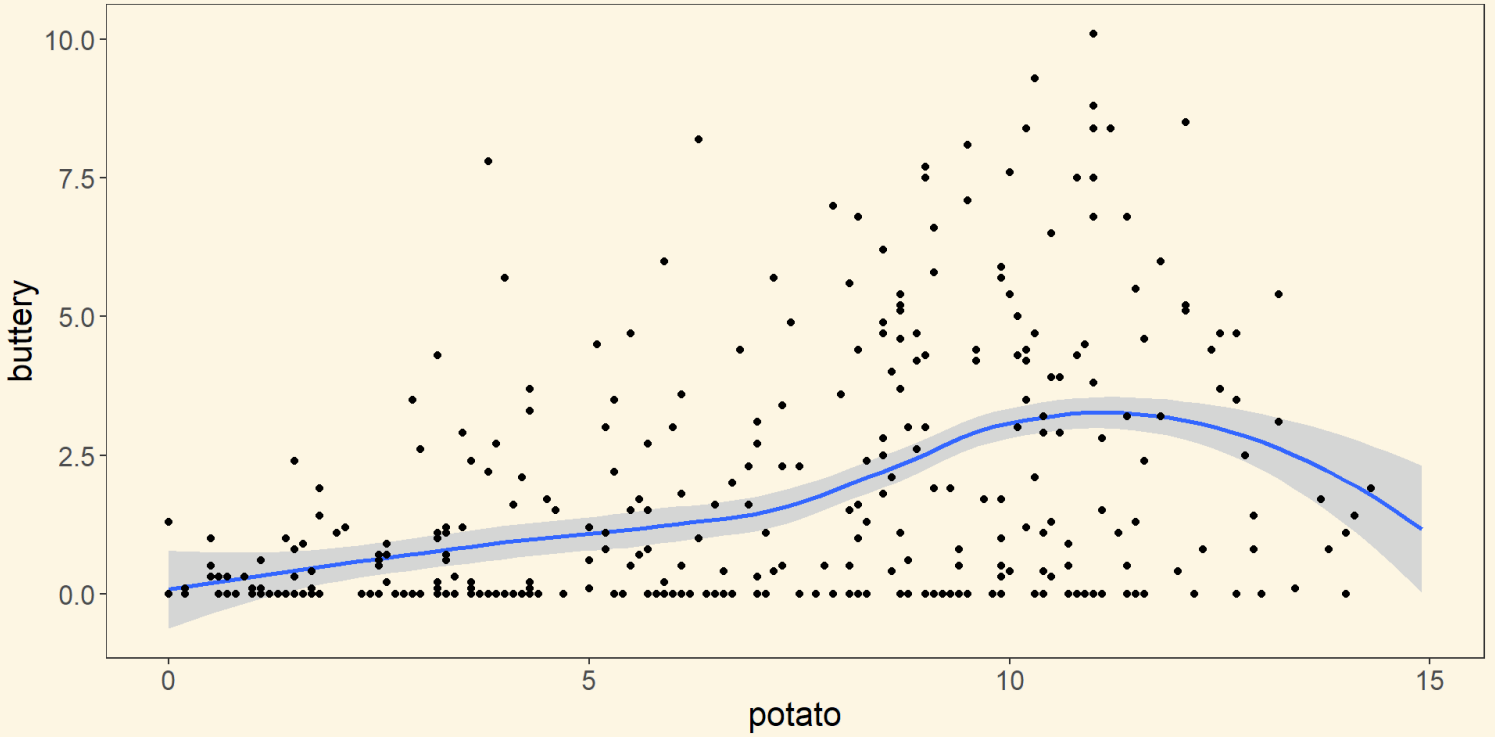


Mappings passed to ggplot are treated as global and apply to all geoms.

DISPLAYING MULTIPLE GEOMS IN A PLOT - FLEXIBILITY

If you set aesthetics globally, you can then override them for a specific layer in the geom call.

```
1 ggplot(french_fries, aes(potato, buttery)) +
2   geom_smooth() +
3   geom_point(data = filter(french_fries, rep == "2"))
```



YOUR TURN

- Try the following code:

```
ggplot(french_fries, aes(treatment, buttery)) +
  geom_boxplot()
```

YOUR TURN - AGAIN

- Try the following code:

```
1 ggplot(french_fries, aes(potato)) +
2   geom_histogram()
```

- Adjust the number of bins using `bins`. What is a good number of bins for this data?
- What does `geom_density()` do with this data?

YOUR TURN - AGAIN

- Try the following code. What's the problem?

```
test_data <- tibble(
  subject = rep(1:4, times = 10),
  condition = rep(1:10, each = 4),
  score = rnorm(40, 0, 1)
)

ggplot(test_data, aes(condition, score, colour = subject)) + geom_point()
```

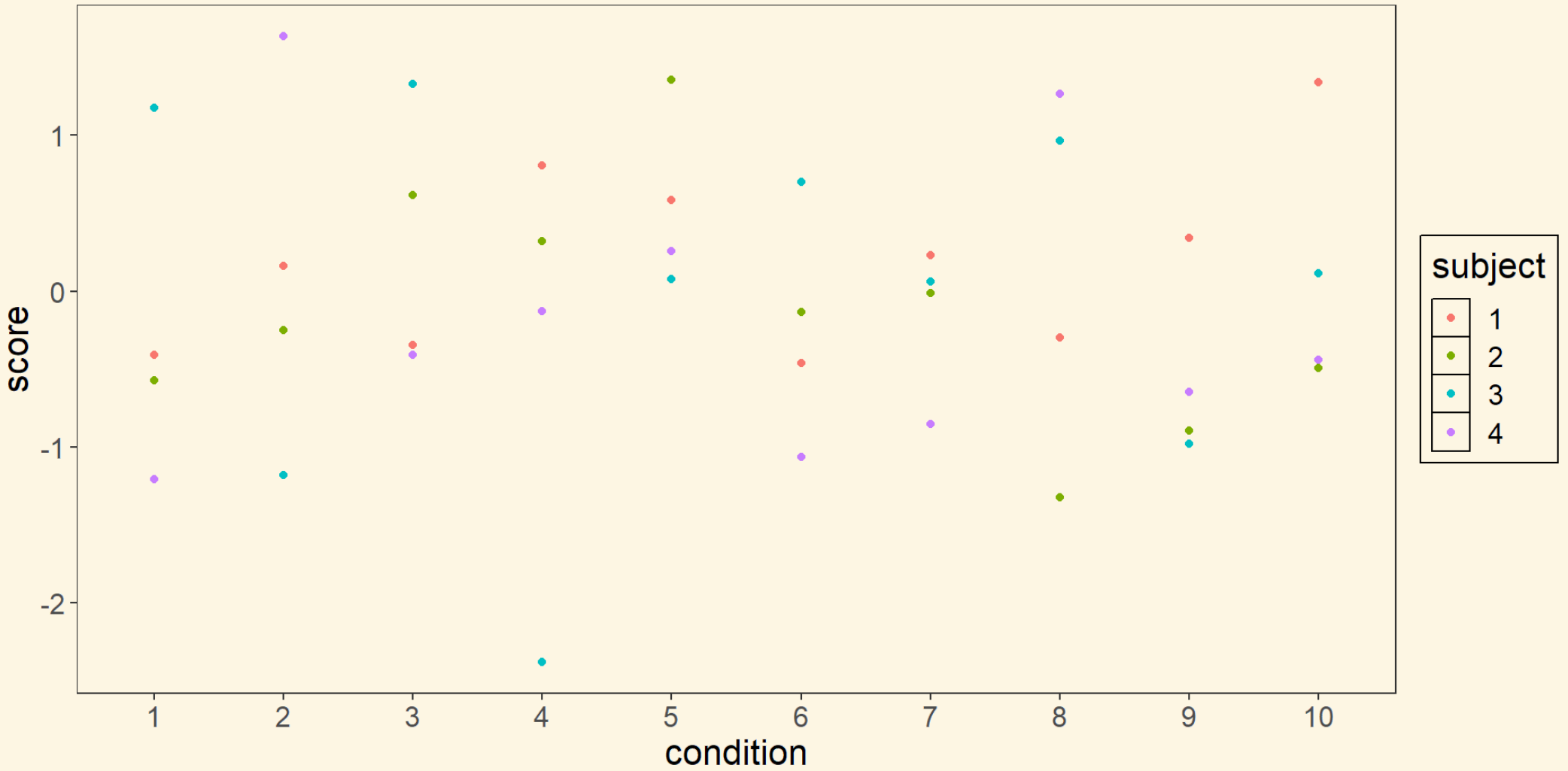
FACTORS

Sometimes we have variables that are **factors** i.e. categorical but are labelled numerically. In this case, we need to tell R explicitly that these variables are factors.

```
test_data <- tibble(
  subject = rep(1:4, times = 10),
  condition = rep(1:10, each = 4),
  score = rnorm(40, 0, 1)
)

test_data <- test_data %>%
  mutate(condition = as_factor(condition),
         subject = as_factor(subject))
```

FACTORS

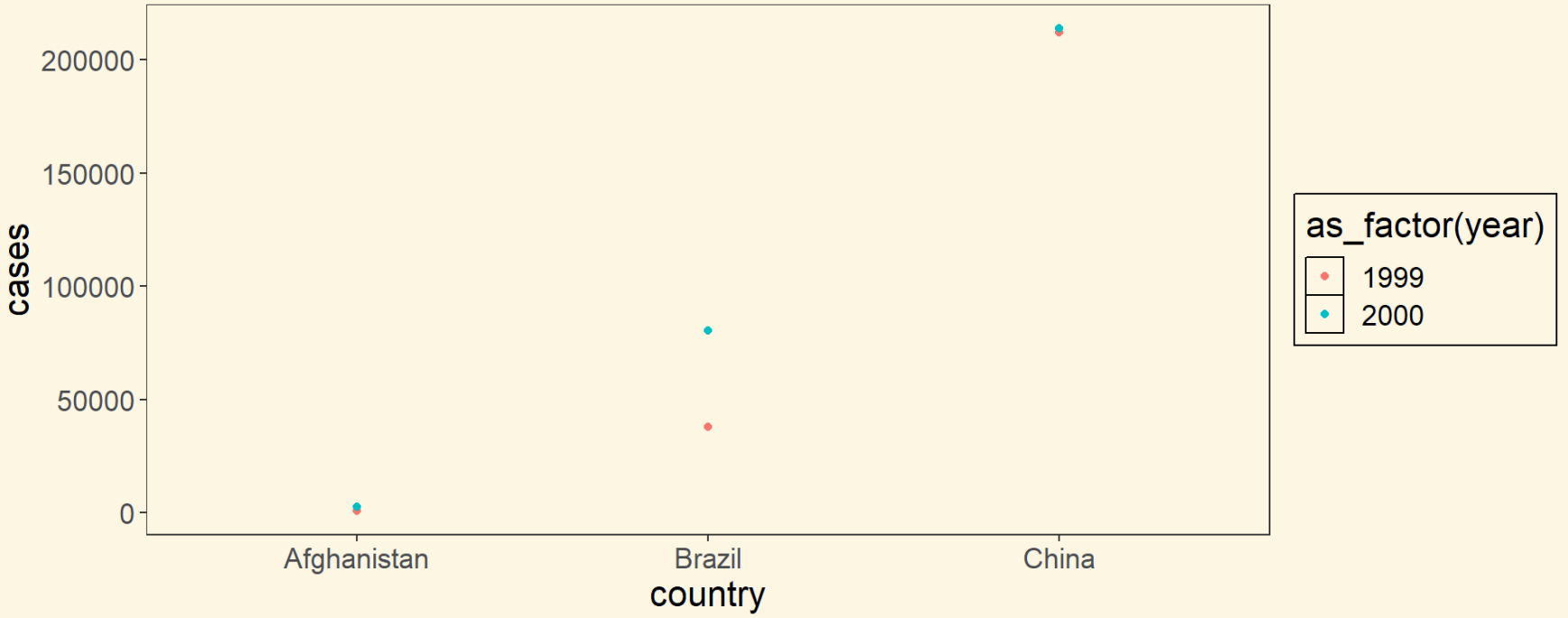
```
1 ggplot(test_data, aes(condition, score, colour = subject)) + geom_point()
```



FACTORS

Sometimes, you might find that you want to re-order your factors e.g. factors are arranged in alphabetical order by default, but you might not want that.

```
1 ggplot(table1, aes(country, cases, colour = as_factor(year))) +
2   geom_point()
```



FACTORS - REORDERING

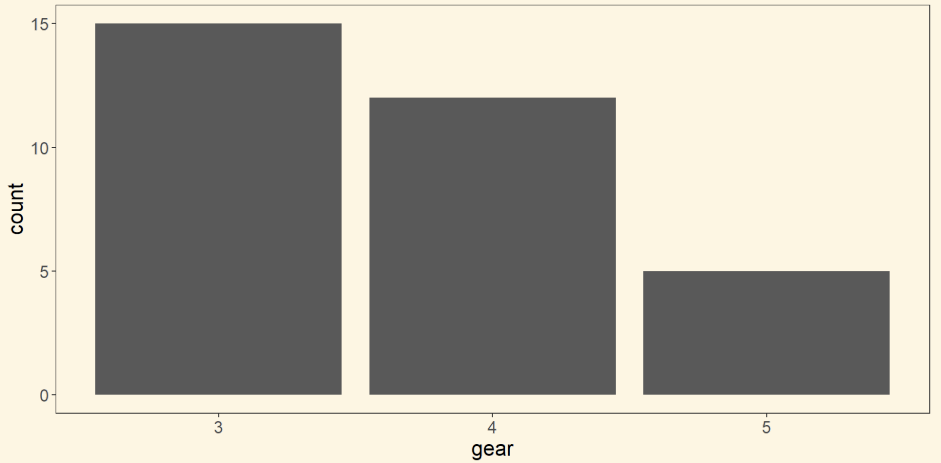
```
1  table1 <- table1 %>%
2    mutate(country = as_factor(country),
3           country = fct_relevel(country, "Brazil", "China", "Afghanistan"),
4           year = as_factor(year))
5
6  ggplot(table1, aes(country, cases, colour = year)) +
7    geom_point()
```

STATISTICAL TRANSFORMATIONS

ggplot doesn't always plot the raw values of your dataset. Sometimes, it calculates new values to plot e.g. geom_bar counts the number of observations that fall into each 'bin'.

Every geom has a default stat, but it is possible to override this if you need.

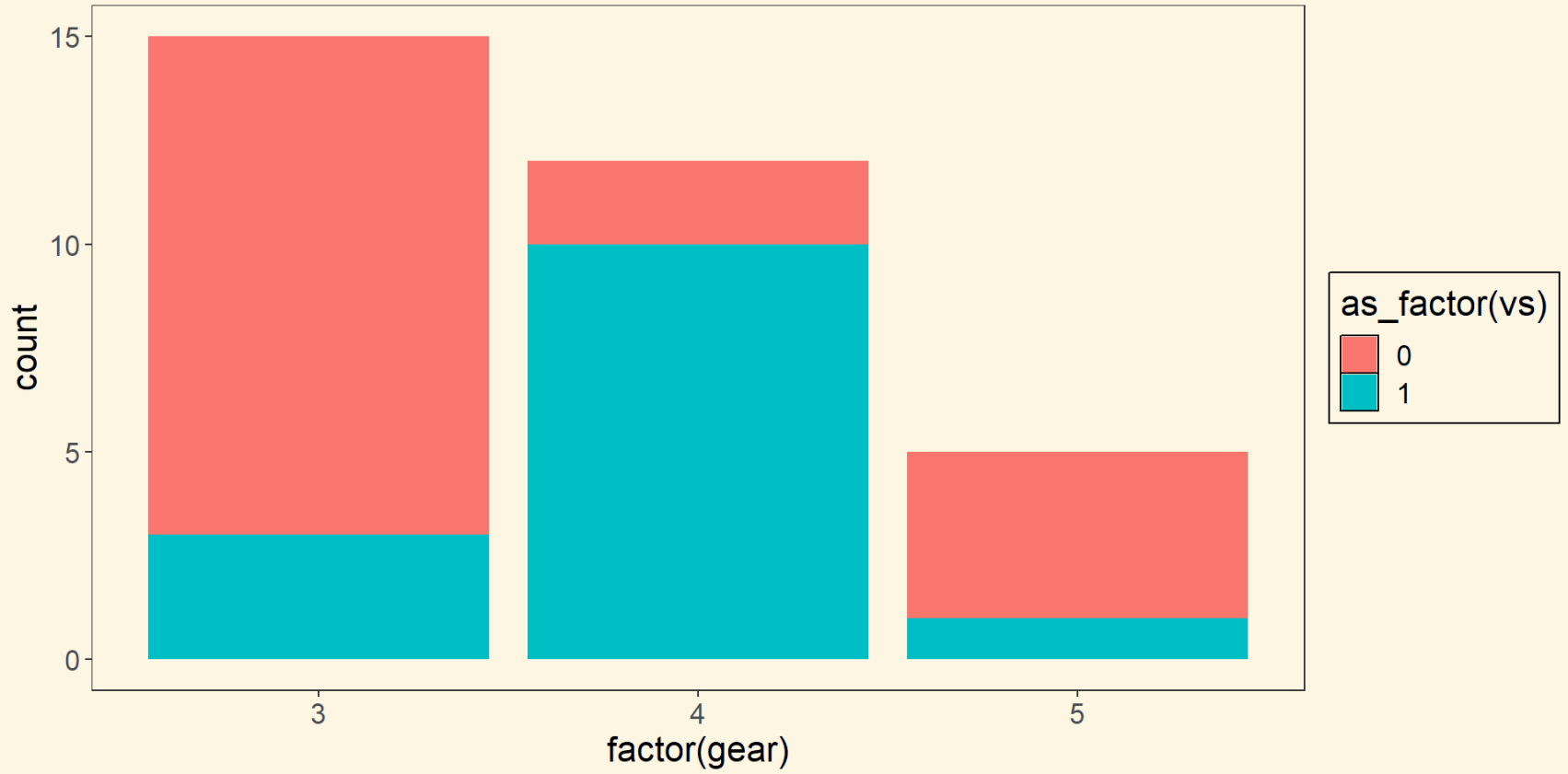
```
1 ggplot(mtcars) +
2   geom_bar(aes(gear))
```



BAR GRAPHS AND THE FILL AESTHETIC

The fill aesthetic gives stacked bar plots…

```
1  ggplot(mtcars) +
2    geom_bar(aes(x = factor(gear),
3             fill = as_factor(vs)))
```



POSITION ADJUSTMENTS - DODGE

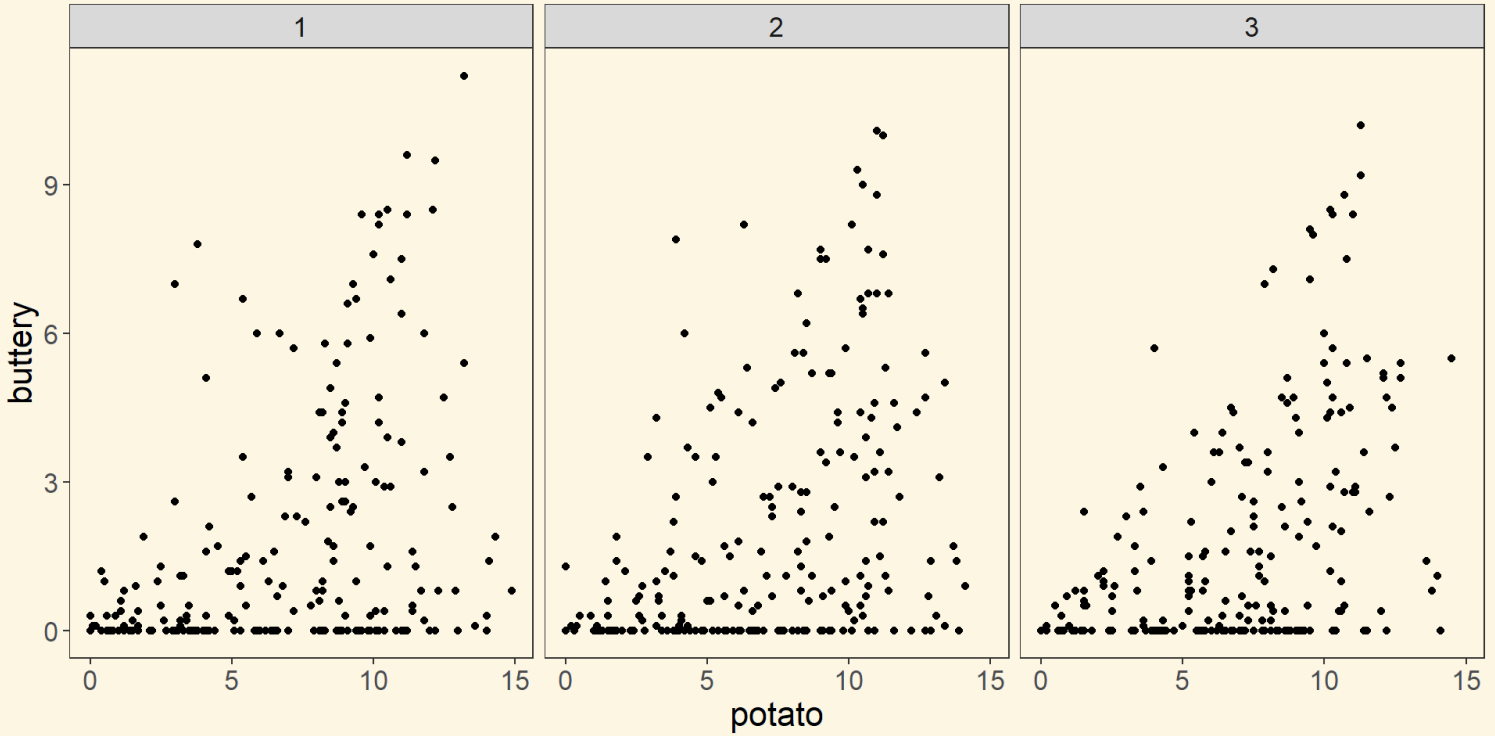
The position aesthetic allows you to select other arrangements: for example, **dodge**, which places the bars side-by-side.

```
ggplot(mtcars) +
  geom_bar(aes(x = factor(gear),
               fill = as_factor(vs)),
               position = "dodge")
```

FACETS

Facets are subplots that each display one subset of data, based on the grouping variables you select.

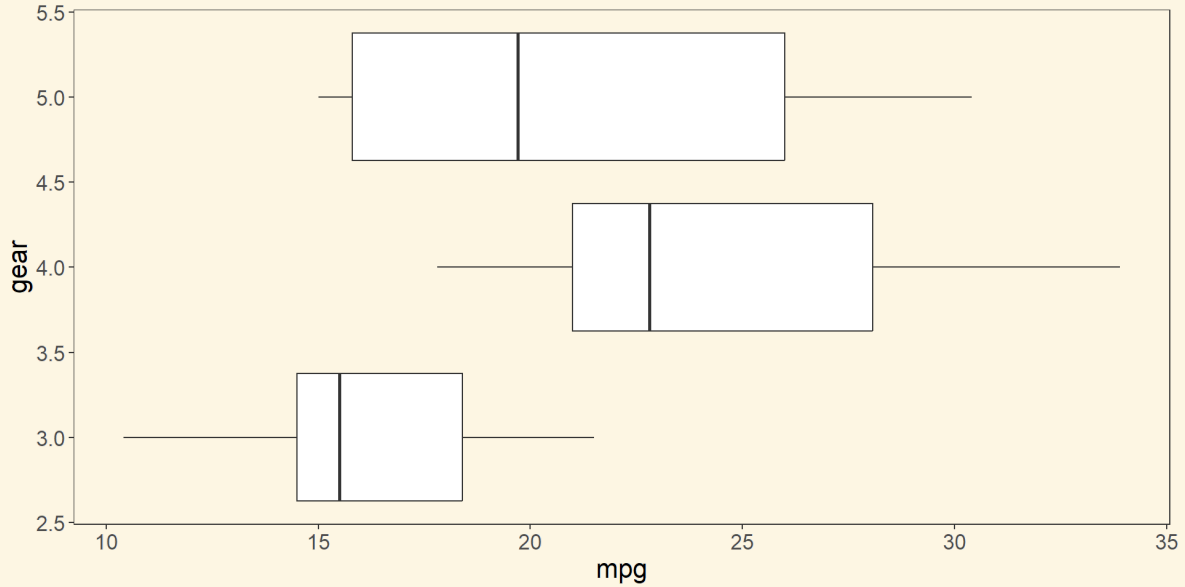
```
ggplot(french_fries) +
  geom_point(aes(potato, buttery)) +
             facet_wrap(~treatment)
```



COORDINATE SYSTEMS

Probably the most generally useful system is coord_flip() which switches the x and y axes (can be useful if you have long treatment names!)

```
1  ggplot(mtcars) +
2    geom_boxplot(aes(gear, mpg,
3                     group = gear)) +
4    coord_flip()
```



THE BASICS OF GGPLOT2: AN OVERVIEW

All ggplot graphs work on the same basic principle:

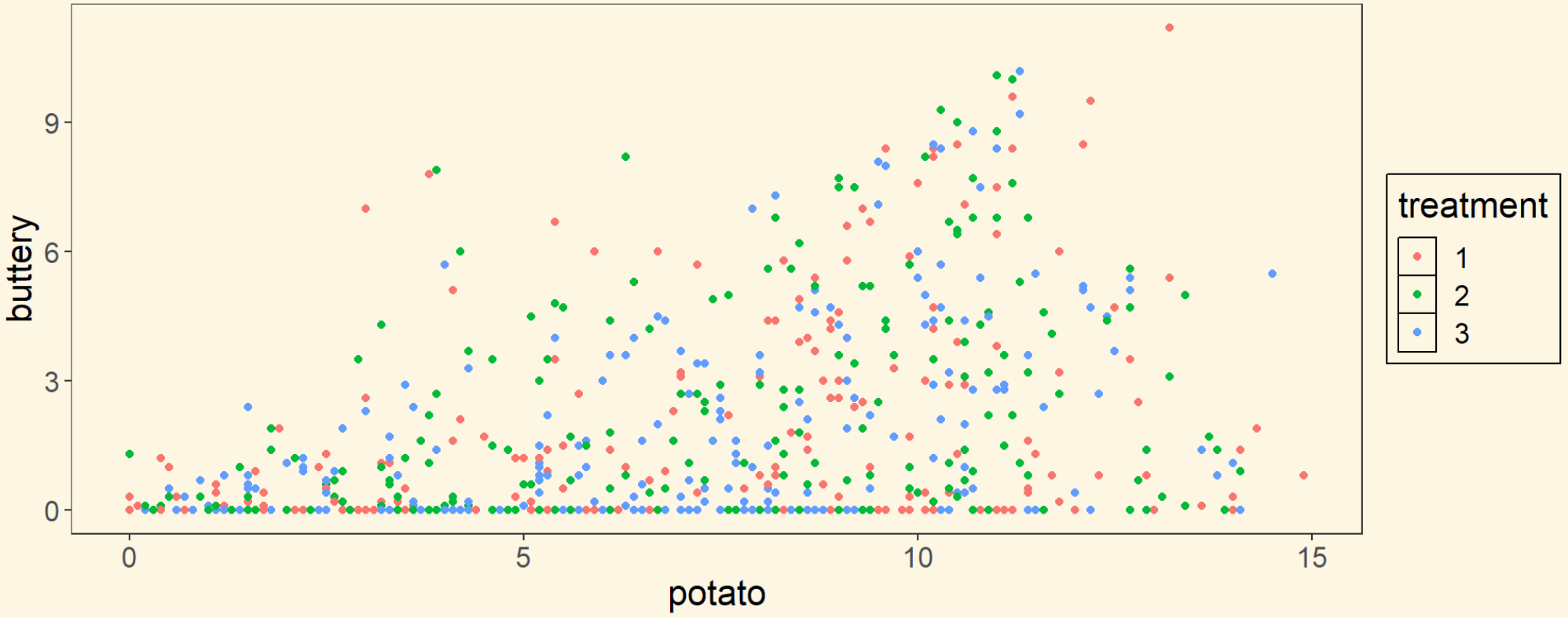
ggplot(data = **DATA**) + **GEOM_FUNCTION**(mapping = aes(**MAPPINGS**), stat = **STAT**, position = **POSITION**) + **COORDINATE_FUNCTION** + **FACET_FUNCTION**

Utilising some/all of this framework should allow you to design a wide range of graphs.

MAKING YOUR GRAPHS LOOK NICER

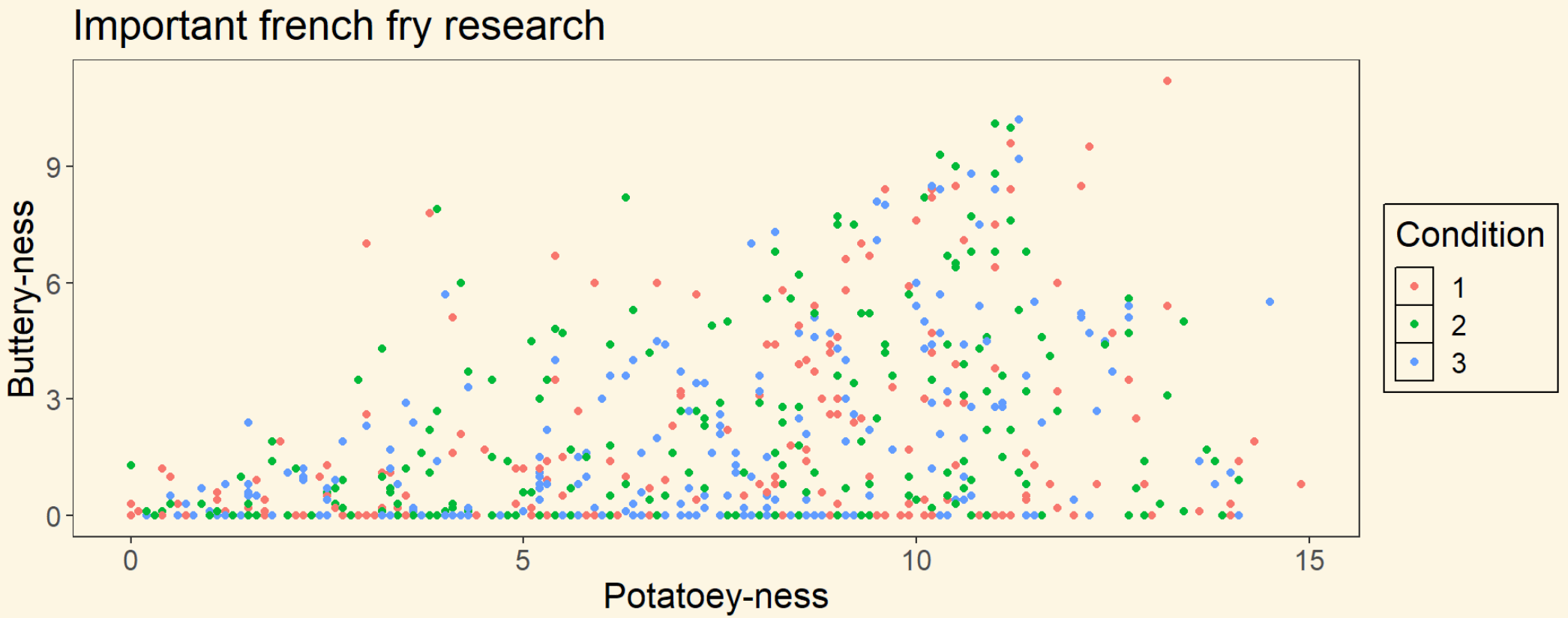
Let's go back to a graph we made earlier. It still looks a bit unfinished.

```
1 ggplot(data = french_fries, aes(potato, buttery, colour = treatment)) +
2   geom_point()
```



ADDING TITLE, LEGEND NAME, AXIS LABELS

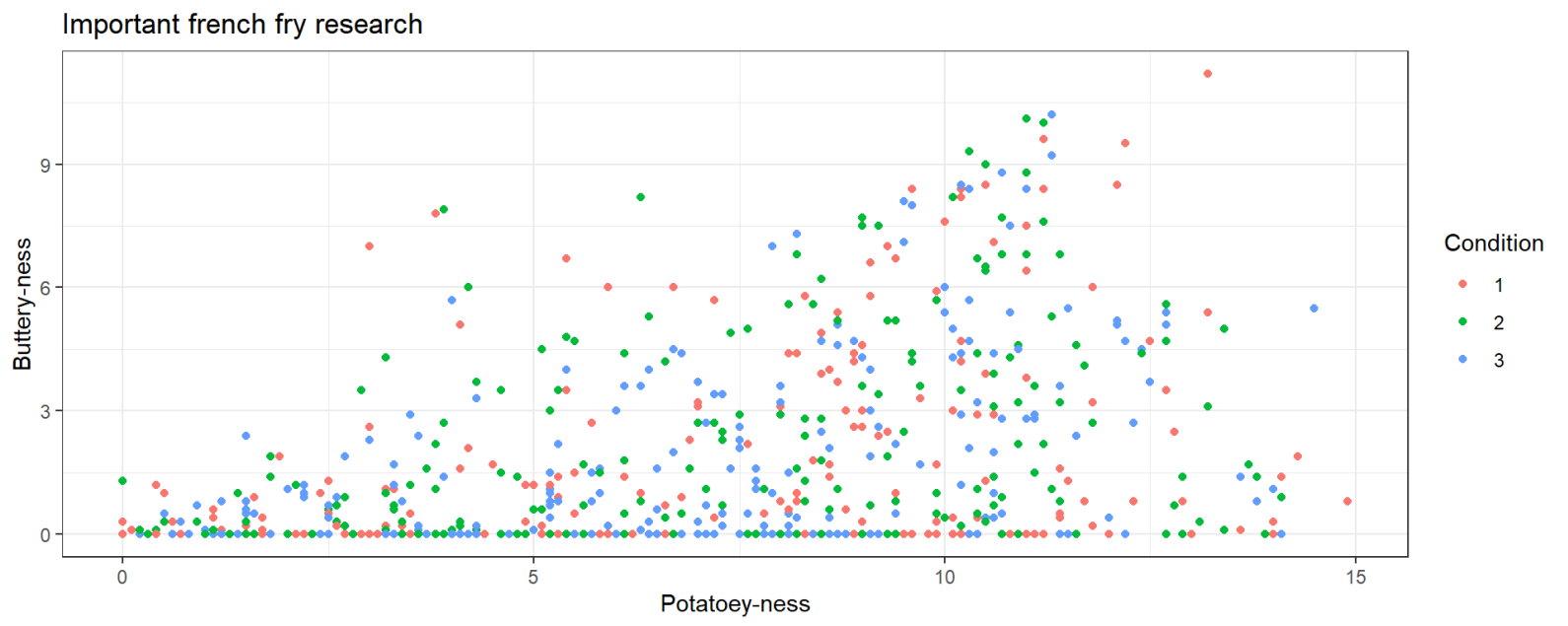
```
1  ggplot(data = french_fries, aes(potato, buttery, colour = treatment)) +
2    geom_point() +
3    xlab('Potatoey-ness') + ylab('Buttery-ness')  +
4    labs(colour = "Condition") + ggtitle('Important french fry research')
```



PICKING A THEME

You can try other themes e.g. `theme_classic` or `theme_light` (or many others… try Googling!)

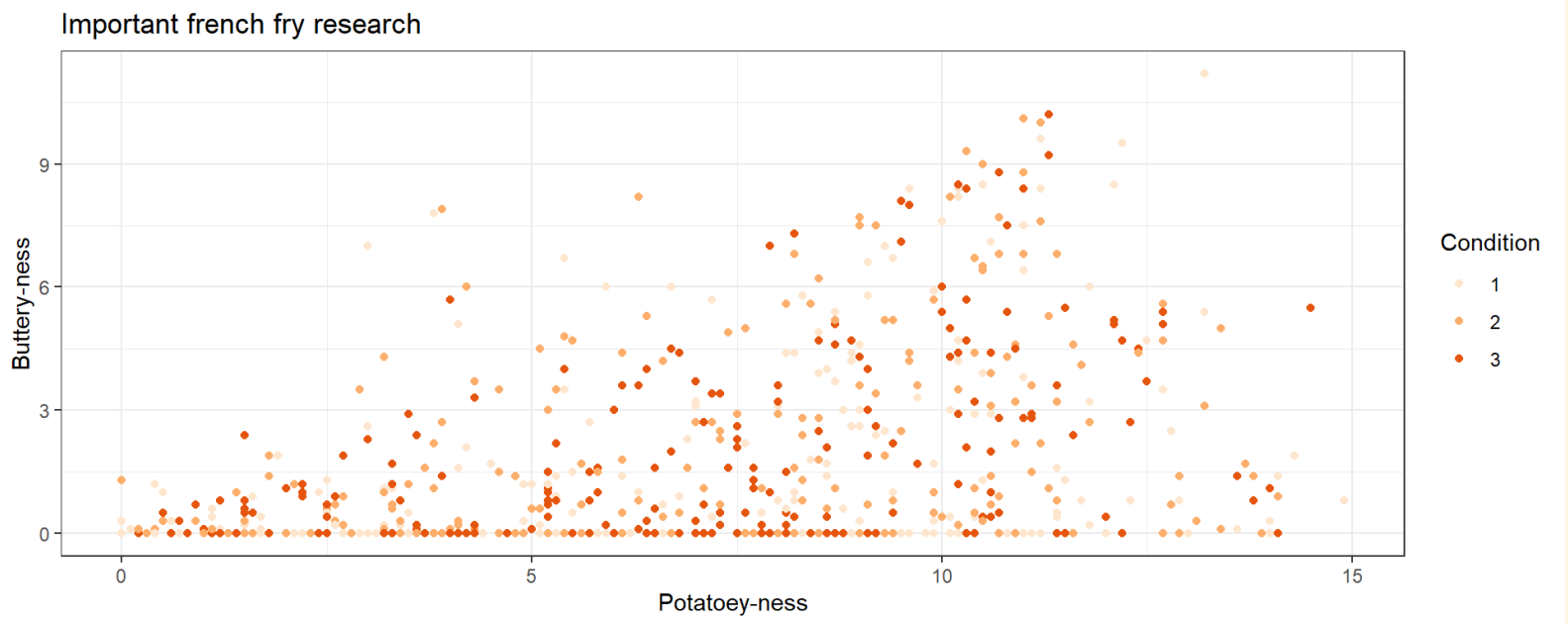
```
ggplot(data = french_fries, aes(potato, buttery, colour = treatment)) +
  geom_point() +
  xlab('Potatoey-ness') + ylab('Buttery-ness')  +
  labs(colour = "Condition") + ggtitle('Important french fry research') +
  theme_bw()
```



CHOOSING COLOURS

R comes with some nice built in colour palettes. Check out `?scale_colour_brewer` in the console to find out more!

```
1  ggplot(data = french_fries, aes(potato, buttery, colour = treatment)) +
2    geom_point() +
3    xlab('Potatoey-ness') + ylab('Buttery-ness')  +
4    labs(colour = "Condition") + ggtitle('Important french fry research') +
5    theme_bw() + scale_colour_brewer(palette = "Oranges")
```



SUMMARY

The key points to take from our sessions about R are:

- R has lots of powerful tools for manipulating your data
- It's good practice to get into the habit of trying not to manipulate your raw data and instead do as much as you can programmatically - that way you know you can't accidentally delete or change any of the raw numbers!
- `ggplot2` provides a 'grammar of graphics' for plotting which allows you to make complex and highly customisable plots
- Again, making graphs in a program such as R means that they are easy to reproduce and modify in the future

FURTHER (FREE!) RESOURCES

Cheat sheets (data visualisation and transformation are particularly useful!): https://www.rstudio.com/resources/cheatsheets/

The R for Data Science book: https://r4ds.hadley.nz/

Data visualisation using ggplot2 book: https://socviz.co/

An introduction to ggplot2: http://www.cookbook-r.com/Graphs/