MONASH University

# Visualising experimental designs with the `edibble` and `deggust` R-packages

Presenter: *Emi Tanaka*

🏛 Department of Econometrics and Business Statistics,
Monash University, Melbourne, Australia

✉ emi.tanaka@monash.edu

🐦 @statsgen

📅 11 Nov 2021 @ Applications of Statistical Procedures in Biological Data

# today's menu

1. Experimental design in *reality*
2. Overview of `edibble`
3. Grammar of graphics with `ggplot2`
4. Visualising experimental designs with `deggust`

These slides made using R powered by HTML/CSS/JS can be found at

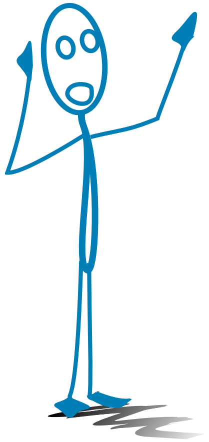
emitanaka.org/slides/stats4bio2021/deggust

①

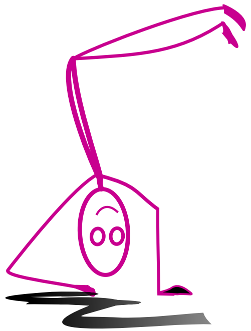
# Experimental design in *reality*

# An experiment generally involves *more than one person*

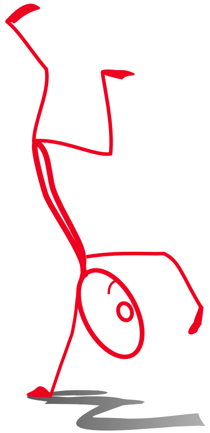
# Meet the cast starring today



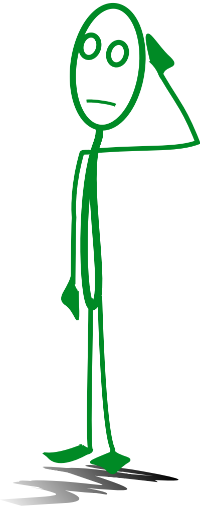
The "**domain expert**"



The "**statistician**"

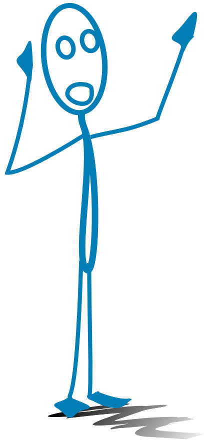


The "**analyst**"



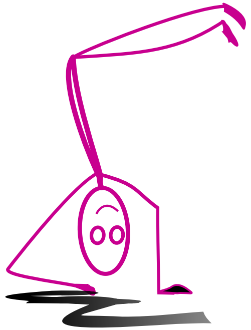
The "**technician**"

Stick person images by OpenClipart-Vectors from Pixabay



The **"domain expert"**

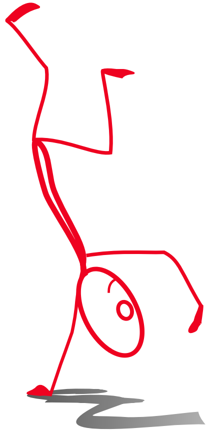
The **"domain expert"** drives the experimental objective and has the intricate knowledge about the subject area



The "**statistician**"

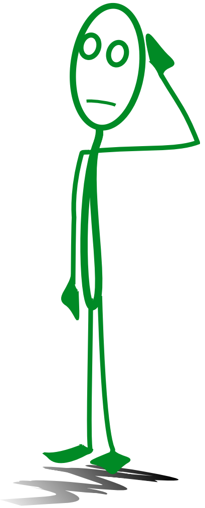
The "**statistican**" creates the experimental design layout after taking into account the statistical and practical constraints.

The "**analyst**" analyses the data after the data is collected.



The "**analyst**"

The **"technician"** carries out the experiment and collects the data.



The **"technician"**

# See the props

The statistical software to design experiments

The software to enter and store data

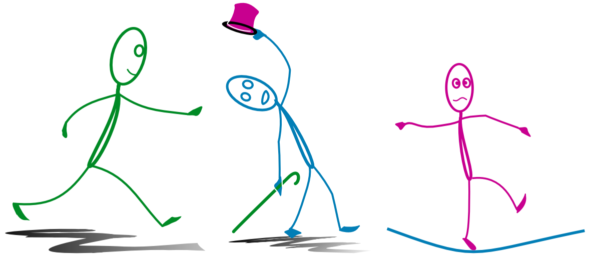
Good old pen and paper

Missing prop: the statistical software to analyse experimental data

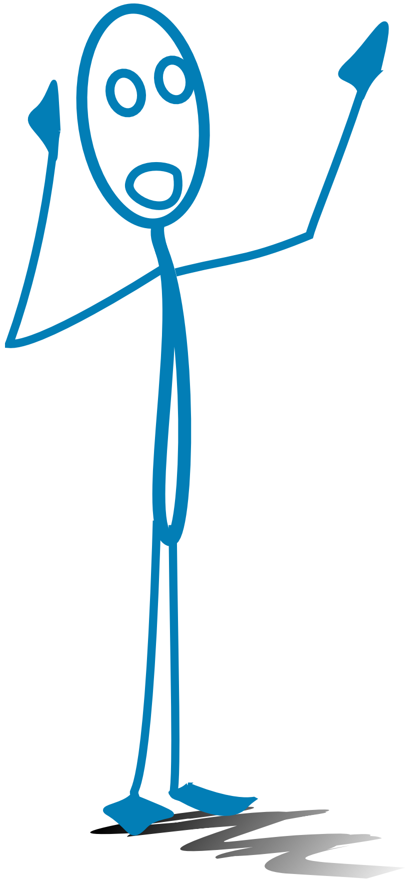
# The actors are purely illustrative.

In practice:

- multiple people can take on each role,
- one person can take on multiple roles, and/or
- a person in the role may not specialise in that role (e.g. the statistician role can be acted out by a non-statistician).

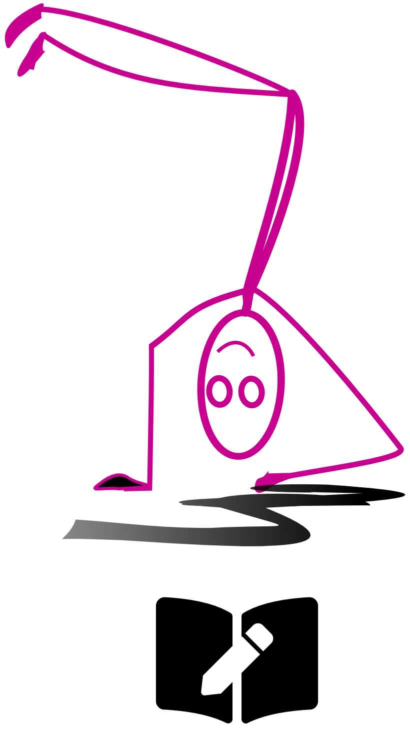


# How we expect experiments are run

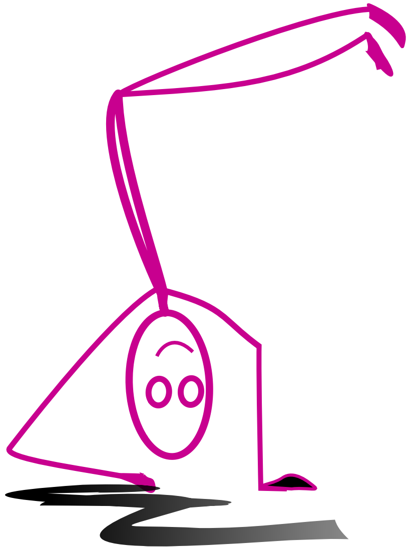


Hey, I need to run an experiment. Can you test irrigation and fertilizer effect on plant growth? Bla bla bla...

Okay. I **got the experimental structure perfectly**. I'll go generate the experimental design layout.

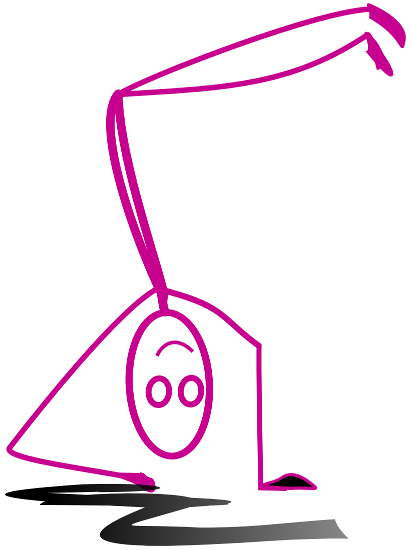


# How we expect experiments are run



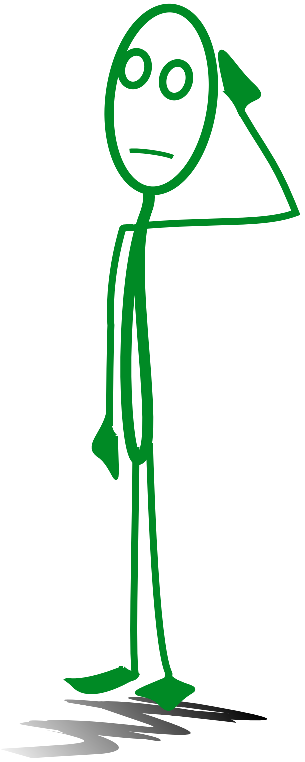
I have a **complete understanding of the experimental structure** so I shall enter it in the software to generate the experimental design

# How we expect experiments are run



Here is the design layout

I'll execute this experiment **exactly as planned** and enter the data with **absolutely no mistake**



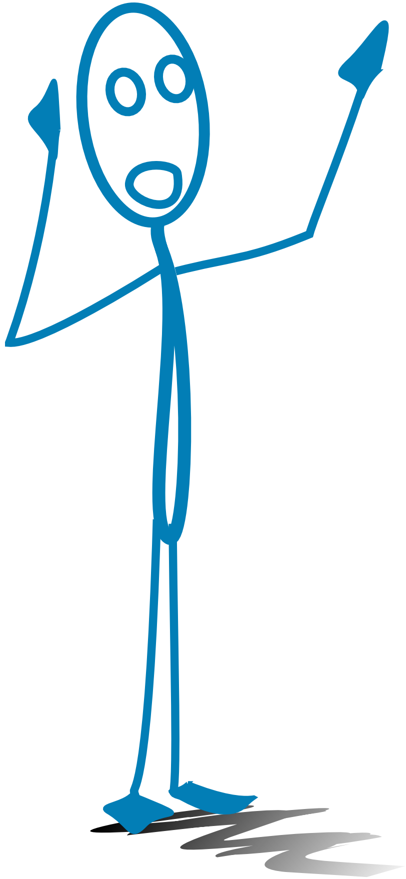
# But

> *communication is complex, fraught with tensions, misunderstandings, and problems — rather than a simple process of creating shared meaning*
>
> — *Littlejohn et al. (2017)*

Littlejohn, Stephen W., Karen A. Foss, and John G. Oetzel. 2017. Theories of Human Communication. Waveland Press, Inc.

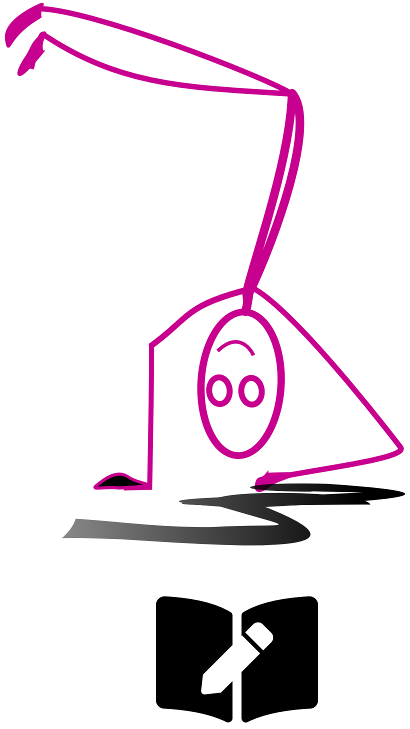
# What happens in reality

- Misunderstandings or incomplete understandings



Hey, I need to run an experiment.
Can you test irrigation and fertilizer
effect on plant growth? Bla bla bla...

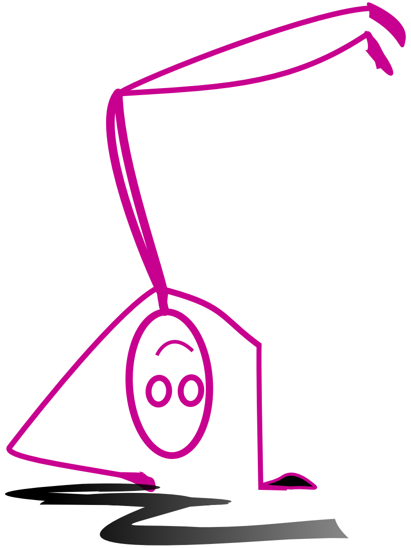
Okay. I think I got it. I'll go see what we
can do for the experimental design.



# What happens in reality

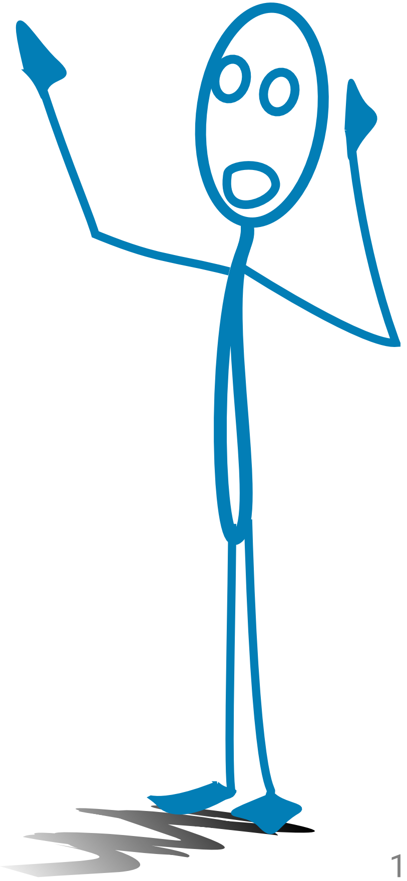
- A lot of back-and-forth to be on the same page

(which is completely natural and okay)



Okay how about this plan.

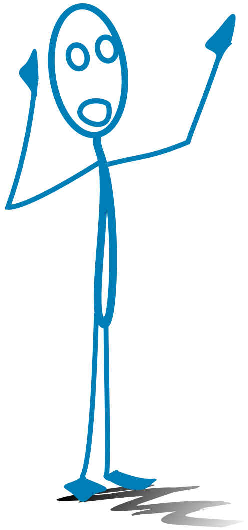
Actually, we can't apply different irrigation methods to these set of plots so we have to change this part. Bla bla.



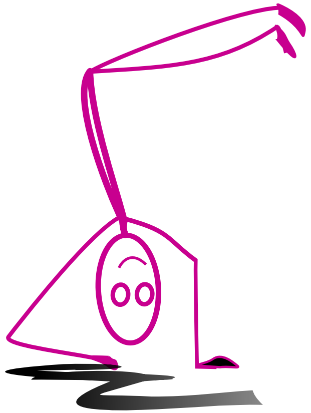
Okay, I'll write this in my notebook.

# What happens in reality

- OR limited communication and decisions made in silo



Oh actually, we didn't have enough seedlings for the test line. Oh well, we'll just leave that empty when we get the plan.

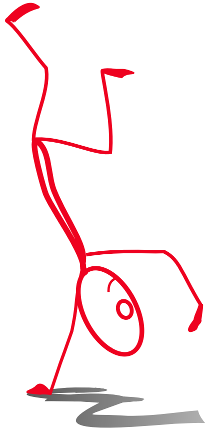


Hmm, I'm not sure about this but I guess this will do for the experimental design.

Oh no! One side of the green house gets more sunlight than the other! Let's move the pots around so that they get the same amount of sunlight.

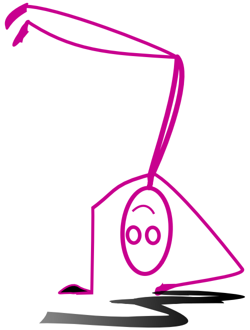
# What happens in reality

- Implicit decisions never explicitly transcribed



Why is the experimental design like this?

I think I had a good reason at the time but I can't remember!

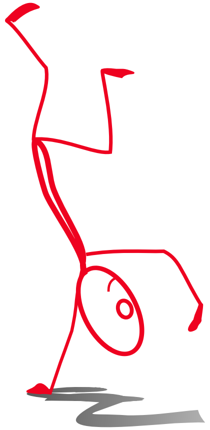


Why are we always on a handstand anyway?

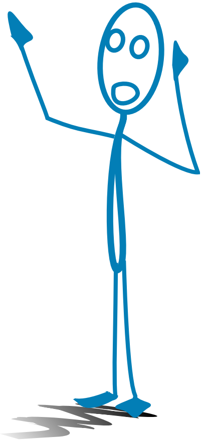
There's no good reason for that!

# What happens in reality

- Knowledge lost



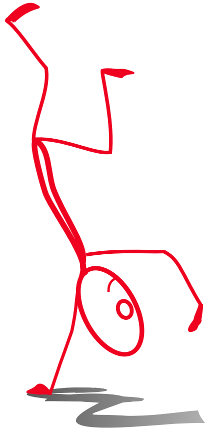
How was the experimental design constructed?



The statistician left a bunch of notes before leaving us for another position but I don't know what's what, so I don't know.

# What happens in reality

- Thinking analysis will save you

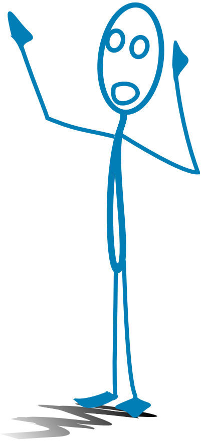


I think we may have data quality issues in this experimental data.

Oh cool data! I can try this fancy statistical model! Hmm, what did you say?

I'm saying that the data may be rubbish.

But I got some numbers from fitting this model.



# Garbage in, garbage out

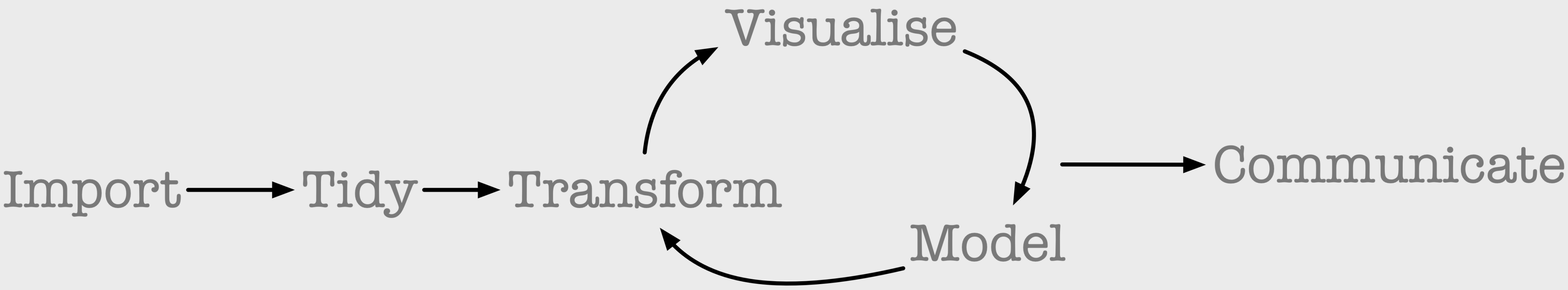
No statistical model, however complex it is, can make any sense of the data if the collected data is rubbish

# Redoing an experiment is *expensive*

There's a higher stake in getting the experiment design wrong compared to getting the analysis wrong.

In some cases, redoing an experiment is not every possible!

# What most statistics (and data science) research is about



Import

Tidy

Transform

Visualise

Model

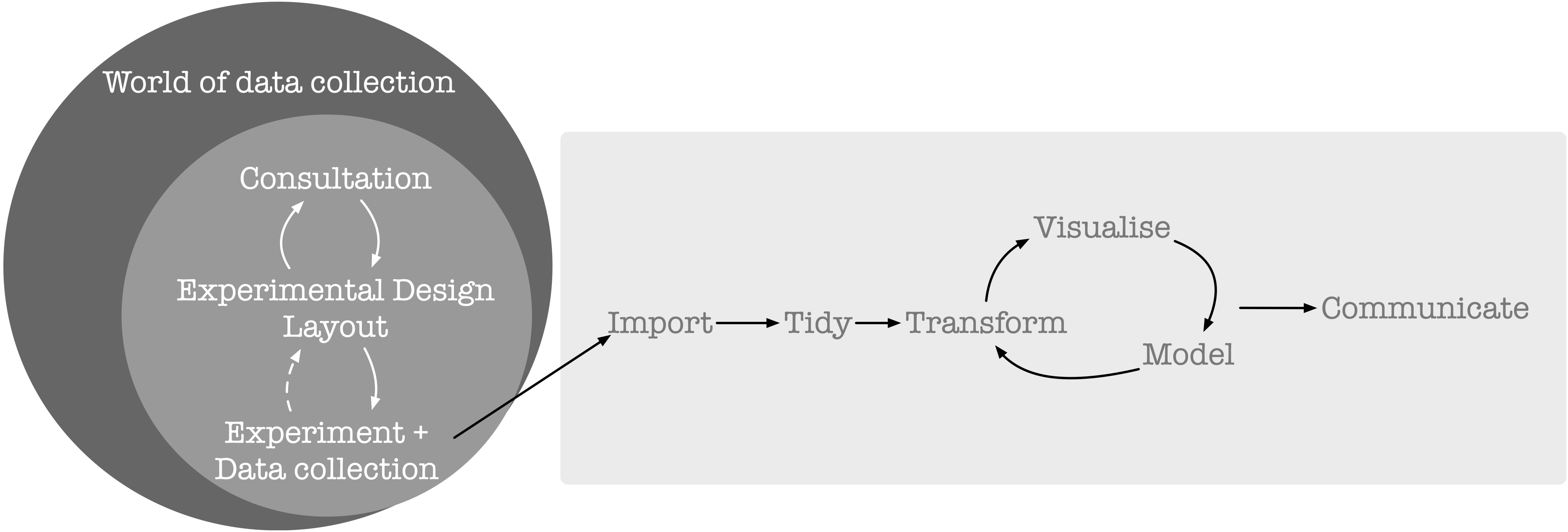
Communicate

(with disproportionate amount in Model)

Grolemund, G., & Wickham, H. (2017). R for Data Science. O'Reilly Media.

# World of data collection

But there is a whole world of data collection before importing data!



- There is probably more gain in extracting information in ensuring the quality of the data than analysis.
- Experimental design research is generally concerned about generating the experimental design layout
- `edibble` aims to complement many great experimental design research to design the whole experiment

②

# Overview of edibble

# The grammar of experimental designs

> An abstract computational framework that maps fundamental experimental components to an object oriented system to build and modify experimental designs. Currently implemented as the `edibble` R-package.

**Package documentation**:

edibble.emitanaka.org

**Source code**:

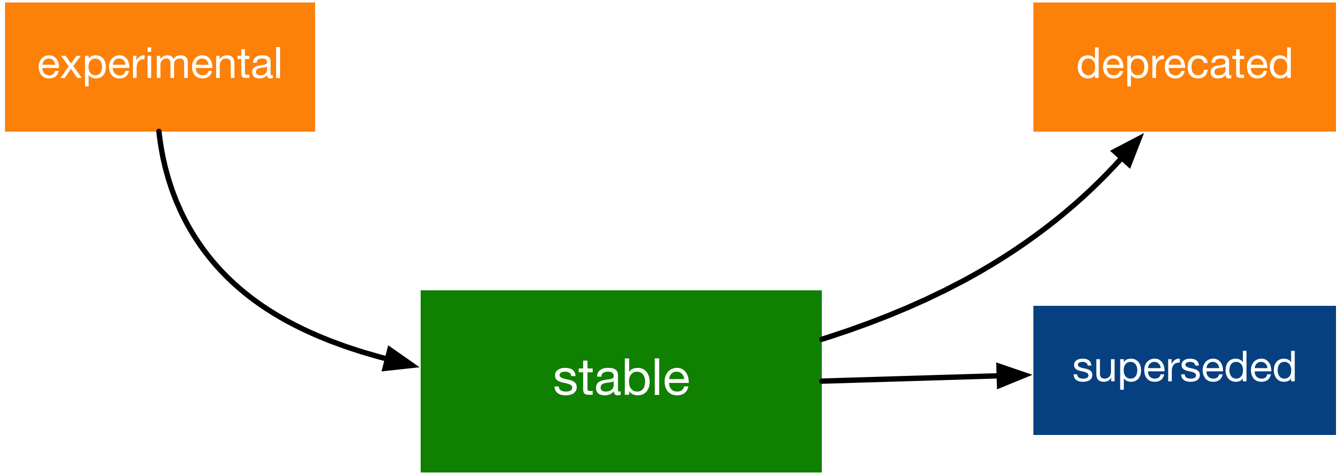
github.com/emitanaka/edibble

**Name origin**: produce **e**xperimental **d**esign table (or `tibble`)

Kirill Müller and Hadley Wickham (2021). tibble: Simple Data Frames. R package version 3.1.5. https://CRAN.R-project.org/package=tibble

# Lifecycle

https://lifecycle.r-lib.org/articles/stages.html



- Currently `edibble` is lifecycle experimental
- Some functions like `allocation_trts` and `randomise_trts` have become lifecycle deprecated in favour of `allot_trts` and `assign_trts`

Lionel Henry and Hadley Wickham (2021). lifecycle: Manage the Life Cycle of your Package Functions. R package version 1.0.1. https://CRAN.R-project.org/package=lifecycle

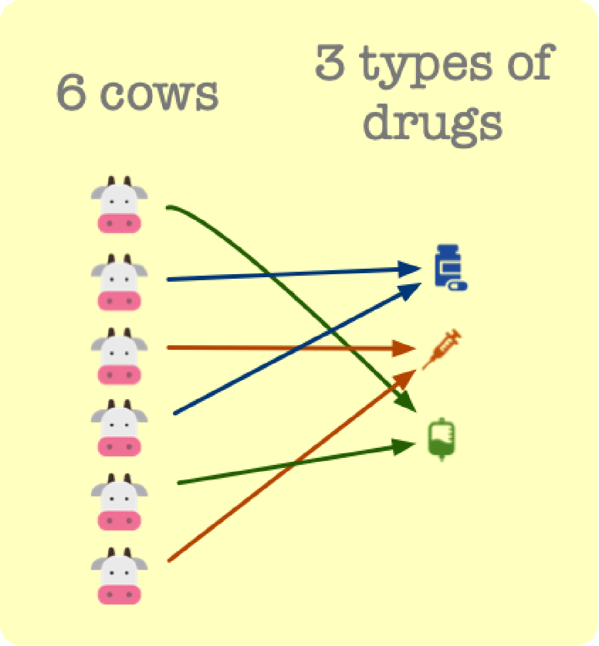
# Reframing how you think about experimental designs

> *Good design considers units and treatments first, and then allocates treatments to units. It does not choose from a menu of named designs.*
>
> *—Rosemary Bailey (2008)*

- `edibble` encourages users to think about designs exactly as Bailey (2008) suggests
- Nevertheless named experimental designs are very prevelant and can be useful to describe particular designs succintly!
- So let's have the best of both worlds

Bailey, R. (2008). Design of Comparative Experiments (Cambridge Series in Statistical and Probabilistic Mathematics). Cambridge: Cambridge University Press. doi:10.1017/CBO9780511611483

# Example ① Completely randomised design

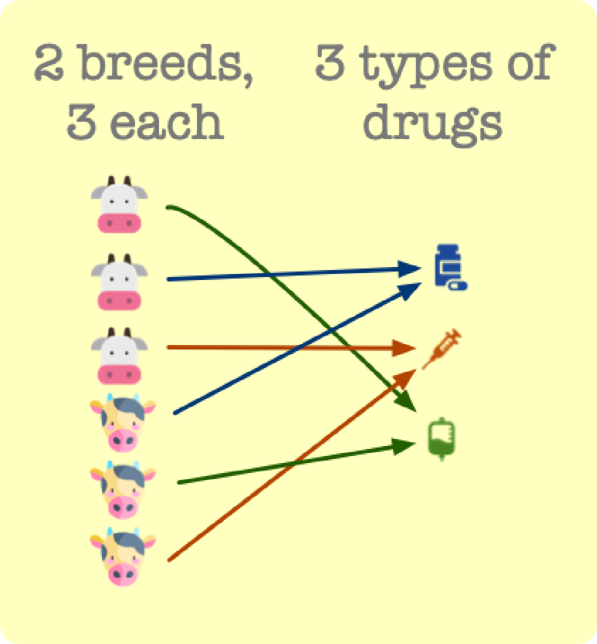


```
library(edibble)
code_classical("crd", t = 3, n = 6)

set.seed(648)
start_design("crd") %>%
  set_units(unit = 6) %>%
  set_trts(treat = 3) %>%
  allot_trts(treat ~ unit) %>%
  assign_trts("random") %>%
  serve_table()
```

- Note: currently only limited named experimental designs are supported
- ⚠ this function name and arguments will likely change in near future

# Example ② Randomised complete block design



```r
library(edibble)
code_classical("rcbd", t = 3, b = 2)

set.seed(231)
start_design("rcbd") %>%
  set_units(block = 2,
            unit = nested_in(block, 3)) %>%
  set_trts(treat = 3) %>%
  allot_trts(treat ~ unit) %>%
  assign_trts("random") %>%
  serve_table()
```

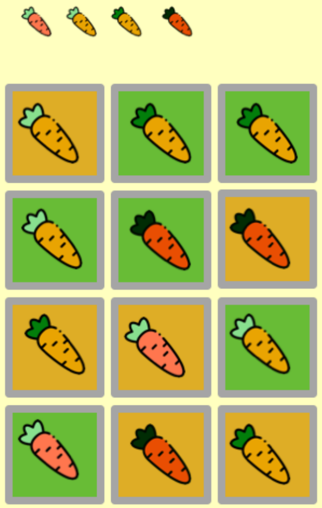
- ⚠ this function name and arguments will likely change in near future

# Example ③ Factorial design

2 types of fertilizer

🟨 🟩

4 varieties of carrots

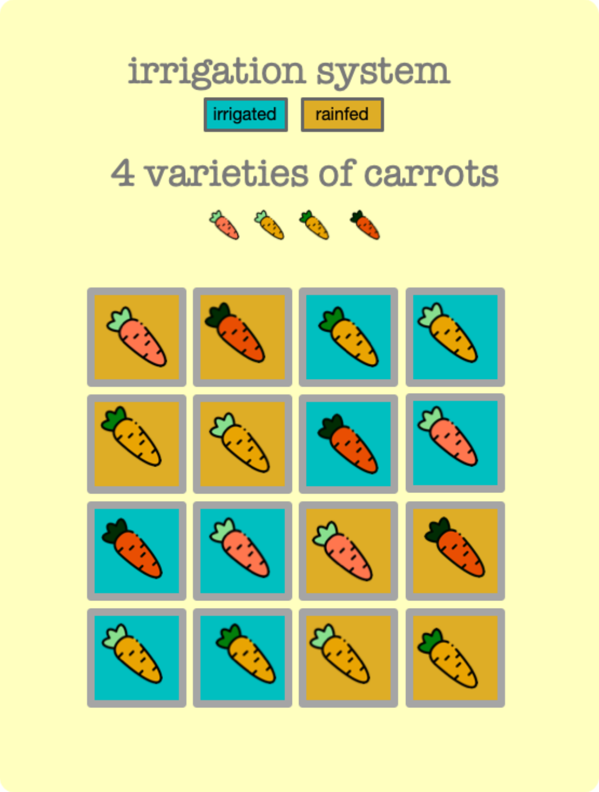


```r
library(edibble)
code_classical("factorial", trt = c(2, 4), n = 16)

set.seed(289)
start_design("factorial") %>%
  set_units(unit = 16) %>%
  set_trts(treat1 = 2,
           treat2 = 4) %>%
  allot_trts( ~ unit) %>%
  assign_trts("random") %>%
  serve_table()
```

- ⚠ this function name and arguments will likely change in near future

# Example ④ Split-plot design

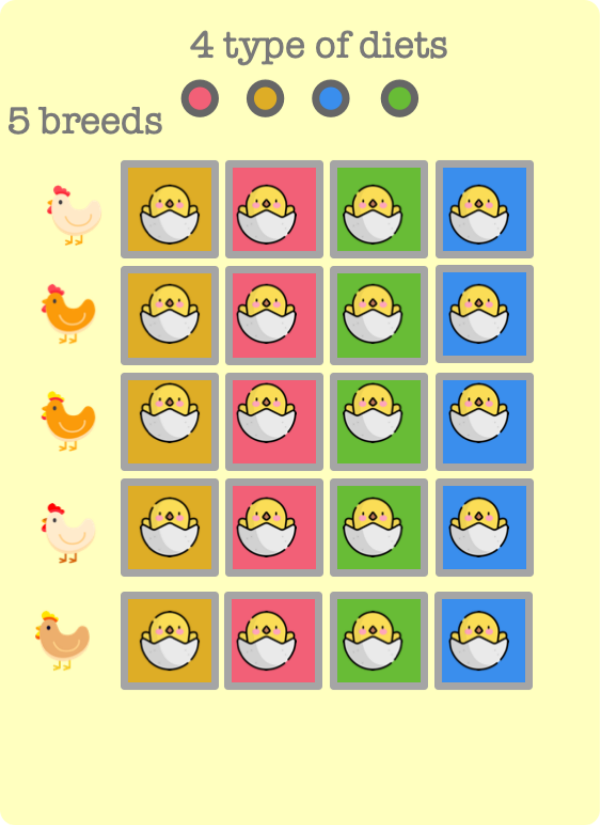


```
library(edibble)
code_classical("split", t1 = 2, t2 = 4, r = 2)

set.seed(566)
start_design("split") %>%
  set_units(mainplot = 4,
            subplot = nested_in(mainplot, 4)) %>%
  set_trts(treat1 = 2,
           treat2 = 4) %>%
  allot_trts(treat1 ~ mainplot,
             treat2 ~ subplot) %>%
  assign_trts("random") %>%
  serve_table()
```

- ⚠ this function name and arguments will likely change in near future

# Example ⑤ Sptrip-plot design



```r
library(edibble)
start_design("Strip-plot") %>%
  set_trts(diet = 4,
           breed = 5) %>%
  set_units(hen = 5,
            order = 4,
            chick = ~hen:order) %>%
  allot_trts(breed ~ hen,
             diet ~ order) %>%
  assign_trts("random") %>%
  serve_table()
```

```
## # An edibble: 20 x 5
##          diet     breed        hen      order       chick
##      <trt(4)> <trt(5)> <unit(5)> <unit(4)> <unit(20)>
##   1     diet1    breed3       hen1     order1      chick1
##   2     diet1    breed4       hen2     order1      chick2
##   3     diet1    breed1       hen3     order1      chick3
```

# Experimental context is important

- Name the variables so it always reminds you of the context

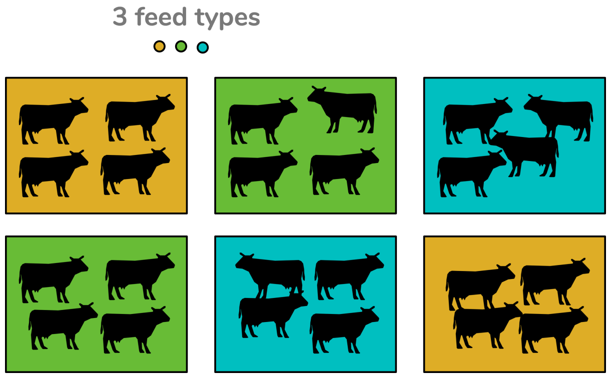
```
start_design("My plant experiment") %>%
  set_units(mainplot = 4,
            subplot = nested_in(mainplot, 4)) %>
  set_trts(water = c("irrigated", "rain-fed"),
           variety = 4) %>%
  allot_trts(water ~ mainplot,
             variety ~ subplot) %>%
  assign_trts("random", seed = 1) %>%
  serve_table()
```

```
## # An edibble: 16 x 4
##      mainplot    subplot     water   variety
##     <unit(4)> <unit(16)>  <trt(2)> <trt(4)>
##   1 mainplot1  subplot1  irrigated variety3
##   2 mainplot1  subplot2  irrigated variety1
##   3 mainplot1  subplot3  irrigated variety4
##   4 mainplot1  subplot4  irrigated variety2
##   5 mainplot2  subplot5  irrigated variety2
##   6 mainplot2  subplot6  irrigated variety4
```

```
start_design("My animal experiment") %>%
  set_units(pen = 4,
            cow = nested_in(pen, 4)) %>%
  set_trts(diet = c("low-card", "high-fat"),
           breed = 4) %>%
  allot_trts(diet ~ pen,
             breed ~ cow) %>%
  assign_trts("random", seed = 1) %>%
  serve_table()
```

```
## # An edibble: 16 x 4
##           pen        cow      diet    breed
##     <unit(4)> <unit(16)> <trt(2)> <trt(4)>
##   1      pen1       cow1  low-card   breed3
##   2      pen1       cow2  low-card   breed1
##   3      pen1       cow3  low-card   breed4
##   4      pen1       cow4  low-card   breed2
##   5      pen2       cow5  low-card   breed2
##   6      pen2       cow6  low-card   breed4
```

# Example ⑥ Calf feeding experiment



```
start_design("Calf feeding") %>%
  set_context(location = "Wagga Wagga") %>%
  set_trts(feed = 3) %>%
  set_units(pen = 6,
            calf = nested_in(pen, 4)) %>%
  allot_trts(feed ~ pen) %>%
  assign_trts("random") %>%
  set_rcrds_of(calf = c("milk", "weight")) %>%
  expect_rcrds(milk = to_be_numeric(with_value(">=", 0)),
               yield = to_be_numeric(with_value(">=", 0))) %>%
  serve_table() %>%
  export_design("calf-design.xlsx", overwrite = TRUE)

## Loading required package: openxlsx

## Wagga Wagga

## ✓ Calf feeding has been written to 'calf-design.xlsx'
```

# Example ❼ Chick weight

- An experiment was conducted on a prairie in Western Canada to find out if insecticides used to control grasshoppers affected the weight of young chicks of ring-necked pheasants, either by affecting the grass around the chicks or by affecting the grasshoppers eaten by the chicks.
- Three insecticides were used, at low and high doses.
- The low dose was the highest dose recommended by the department of agriculture; the high dose was four times as much as the recommended dose, to assess the effects of mistakes.
- The experimental procedure took place in each of three consecutive weeks.
- On the first day of each week a number of newly-hatched female pheasant chicks were placed in a brooder pen.
- On the third day, the chicks were randomly divided into twelve groups of six chicks each.
- Each chick was given an identification tape and weighed.
- On the fourth day, a portion of the field was divided into three strips, each of which was divided into two swathes.
- The two swathes within each strip were sprayed with the two doses of the same insecticide.
- Two pens were erected on each swathe, and one group of pheasant chicks was put into each pen.
- For the next 48 hours, the chicks were fed with grasshoppers which had been collected locally.
- Half the grasshoppers were anaesthetized and sprayed with insecticide; the other half were also anaesthetized and handled in every way like the first half except that they were not sprayed.
- All grasshoppers were frozen.
- The experimenters maintained a supply of frozen grasshoppers to each pen, putting them on small platforms so that they would not absorb further insecticide from the grass.
- In each swathe, one pen had unsprayed grasshoppers while the other had grasshoppers sprayed by the insecticide which had been applied to that swathe.
- At the end of the 48 hours, the chicks were weighed again individually.

😵 😵 😵

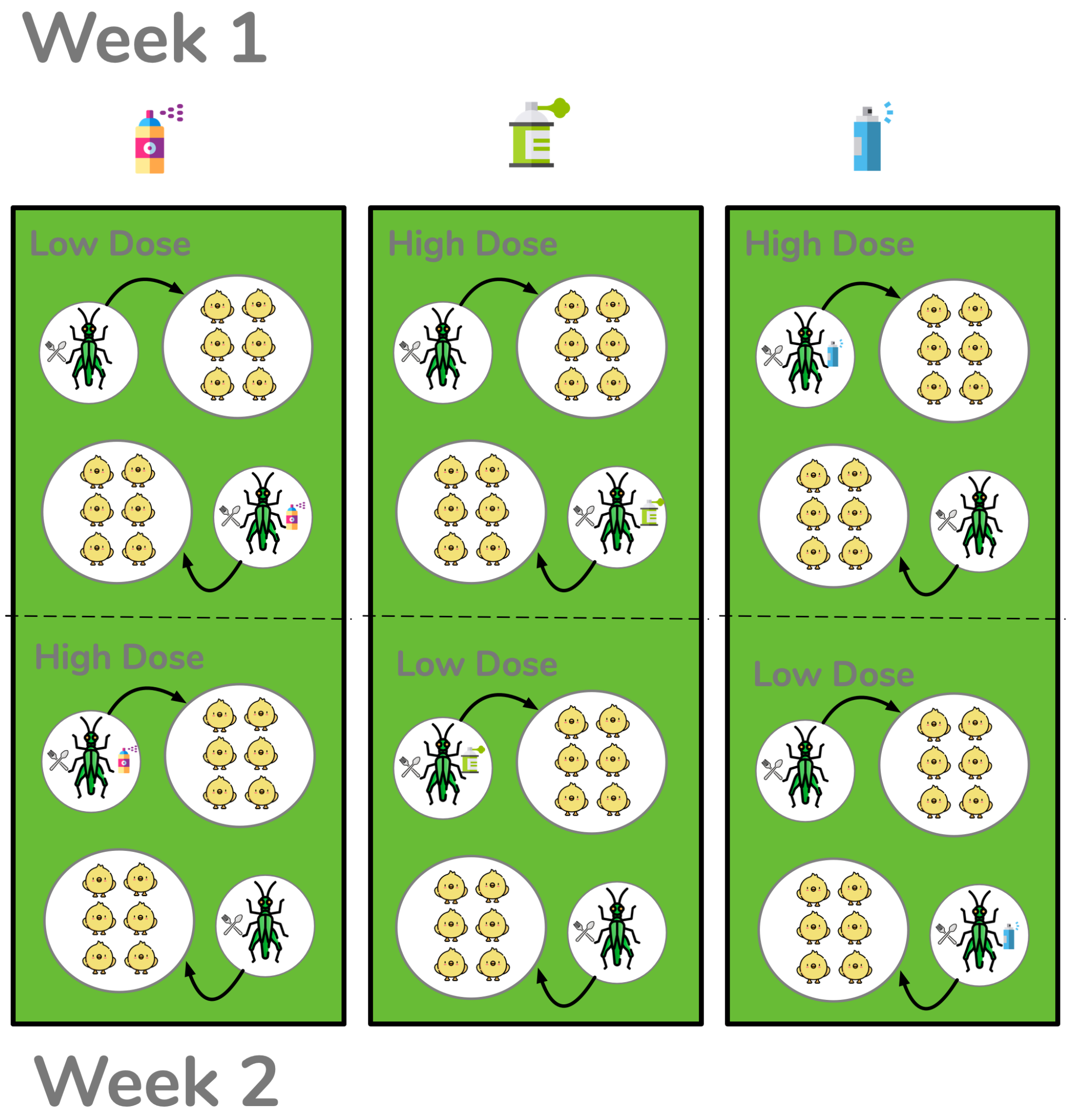
Experiment based from Martin et al. (1996) Effects of grasshopper-control insecticides on survival and brain acetylcholinesterase of pheasant (Phasianus colchicus) chicks. Environmental Toxicology and Chemistry **15**(4) 518-524.

# Example ⑦ Chick weight: the edibble code

```
start_design("Chick weight") %>%
  set_trts(insecticide = 3,
           dose = c("low", "high")) %>%
  set_units(week = 3,
            strip = nested_in(week, 4),
            swathes = nested_in(strip, 2),
            pen = nested_in(swathes, 2),
            chick = nested_in(pen, 6)) %>%
  set_trts(food = c("spray", "no-spray")) %>%
  allot_trts(insecticide ~ strip,
             dose ~ swathes,
             food ~ pen) %>%
  assign_trts("random") %>%
  set_rcrds(weight = chick) %>%
  serve_table()
```

```
## # An edibble: 288 x 9
##      insecticide     dose      week      strip    swathes        pen
```

# Example ⑦ Chick weight: closer look



Skeleton ANOVA

| Stratum | Source | df |
|---|---|---|
| Week | Week | 2 |
| Strips | Insecticide | 2 |
| | Strips Residual | 4 |
| Swathes | Dose | 1 |
| | Insecticide : Dose | 2 |
| | Swathes Residual | 6 |
| Pens | Food | 1 |
| | Insecticide : Food | 2 |
| | Dose : Food | 1 |
| | Insecticide : Dose : Food | 2 |
| | Pens Residual | 12 |
| Chicks (OU) | OU Residual | 180 |

# Example 8 Unbalanced designs

All designs thus far have been **balanced** (i.e. equal replicate) and **complete** (each treatment appears the same number of times in each block)...

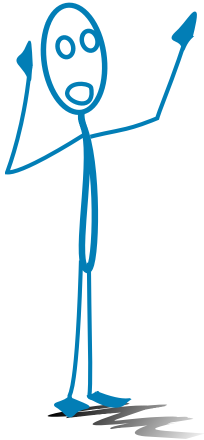
## What about unbalanced and/or incomplete designs?

Reference level by its name:

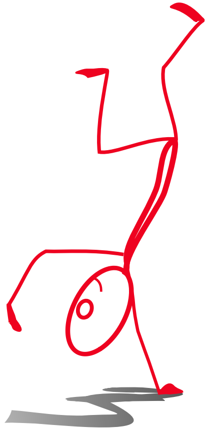
```
start_design("unbalanced & incomplete") %>%
  set_units(site = c("Horsham", "Narrabri",
                     "Wagga", "Roseworthy"),
            plot = nested_in(site,
                             "Horsham" ~ 6,
                             "Narrabri" ~ 3,
                             . ~ 4)) %>%
  set_trts(breed = 4) %>%
  allot_trts(breed ~ plot) %>%
  assign_trts("random") %>%
  serve_table()
```

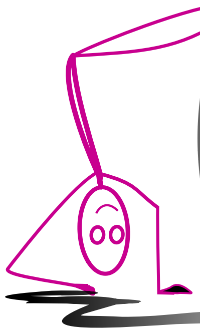
Reference level by number:

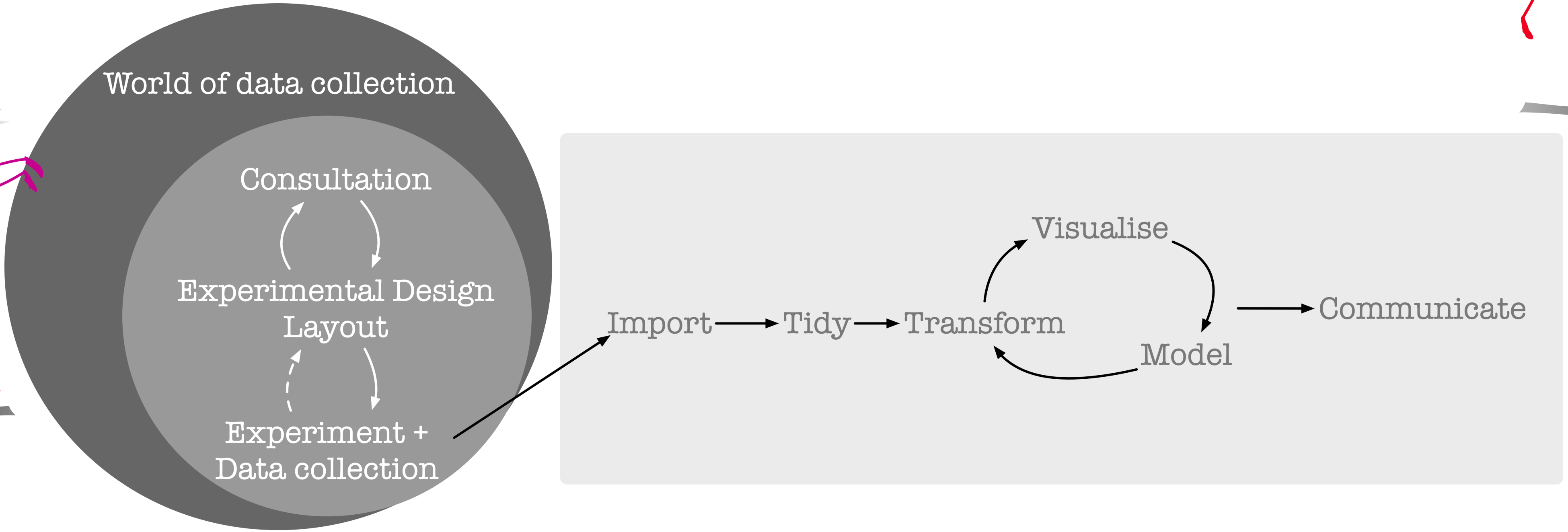
```
start_design("unbalanced & incomplete") %>%
  set_units(site = c("Horsham", "Narrabri",
                     "Wagga", "Roseworthy"),
            plot = nested_in(site,
                             1 ~ 6,
                             2 ~ 3,
                             . ~ 4)) %>%
  set_trts(breed = 4) %>%
  allot_trts(breed ~ plot) %>%
  assign_trts("random") %>%
  serve_table()
```



> “
>
> *communication is complex, fraught with tensions, misunderstandings, and problems — rather than a simple process of creating shared meaning*
>
> — *Littlejohn et al. (2017)*







# What can help to communicate more effectively?

③

# Grammar of graphics with `ggplot2`

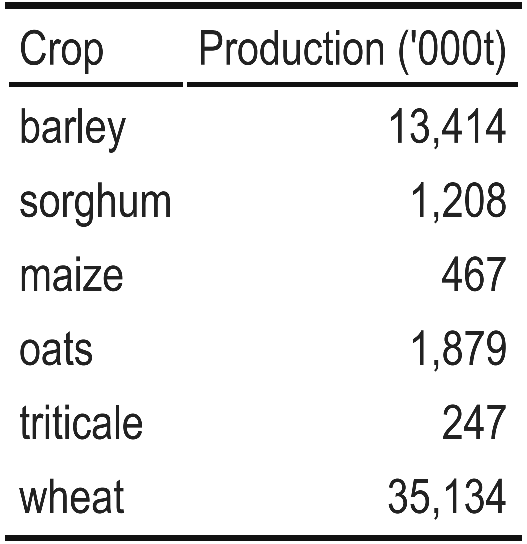
# Grammar of graphics: origin and implementations

- Initial instances of the grammar of graphics was mentioned by William C. Brinton
- A full computational framework was developed by Leland Wilkinson with implementation in **SYSTAT**.
- An interpretation of the grammar of graphics by Hadley Wickham (as part of his PhD, 2008) was implemented in **R** as the `ggplot2` package.
- Emulation of `ggplot2` in **python** started to be developed:
  - `ggpy` (defunct)
  - `plotnine` by Hassan Kibirige,
  - `seaborn` by Michael Waskom (this one is not quite trying to emulate ggplot)
- In **Julia**, `Gadfly` by Daniel C. Jones implements the grammar of graphics.
- In **Matlab**, `gramm` by Pierre Morel implements the grammar as a toolbox.
- In **Javascript**, `G2` by AntV team, adding also interactivity, with this version emulated in R as `g2r` package.
- `ggplot2` is arguably the most popular interpretation of grammar of graphics with over 35,000 citations

Brinton (1914) Graphic methods for presenting facts
Wilkinson (1999) The Grammar of Graphics. *Statistics and Computing. Springer, 1st edition.*

# Data ① Crop production in Australia

In Australia, total production of each crop nationally are



| Crop | Production ('000t) |
|---|---:|
| barley | 13,414 |
| sorghum | 1,208 |
| maize | 467 |
| oats | 1,879 |
| triticale | 247 |
| wheat | 35,134 |

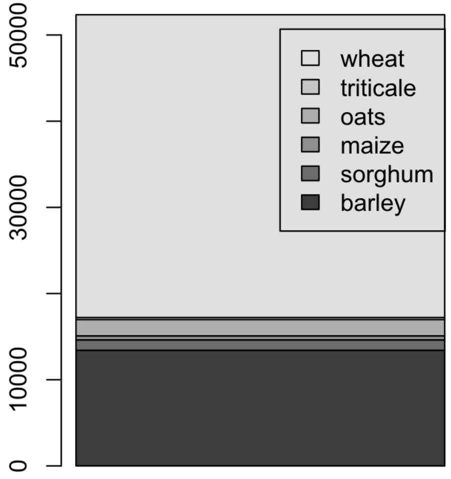
Data source: ABARES Australian Crop Report, 2017, No. 181

# Plotting with "base R"

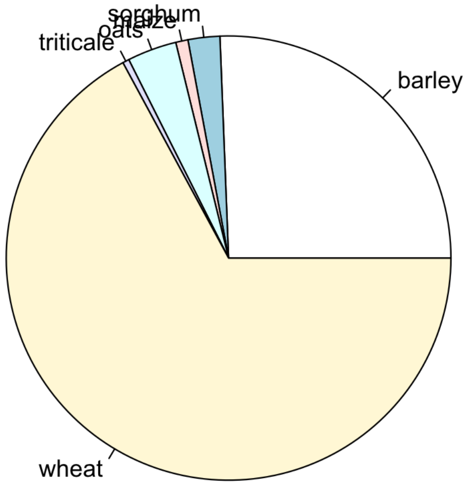
```
df
```

```
##         crop  prod
## 1     barley 13414
## 2    sorghum  1208
## 3      maize   467
## 4       oats  1879
## 5 triticale   247
## 6      wheat 35134
```

```
barplot(as.matrix(df$prod),
        legend = df$crop)
```



```
pie(df$prod, labels = df$crop)
```



Single purpose functions to generate "named plots"

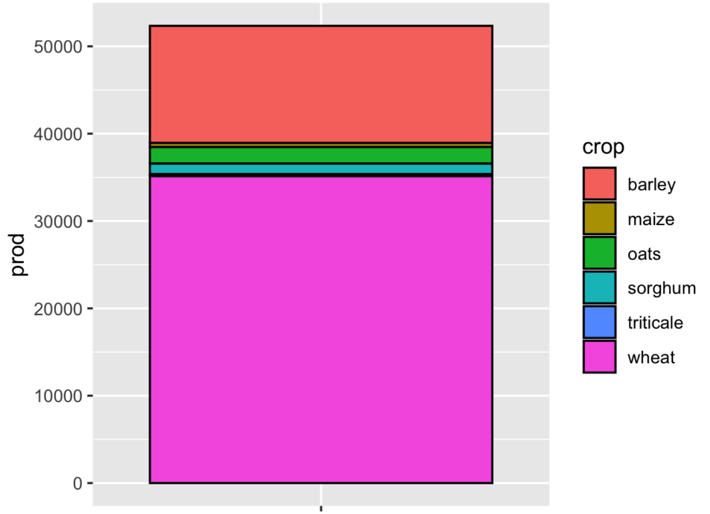
R Core Team (2020) R: A Language and Environment for Statistical Computing https://www.R-project.org/

# Plotting with the ggplot2 R-package

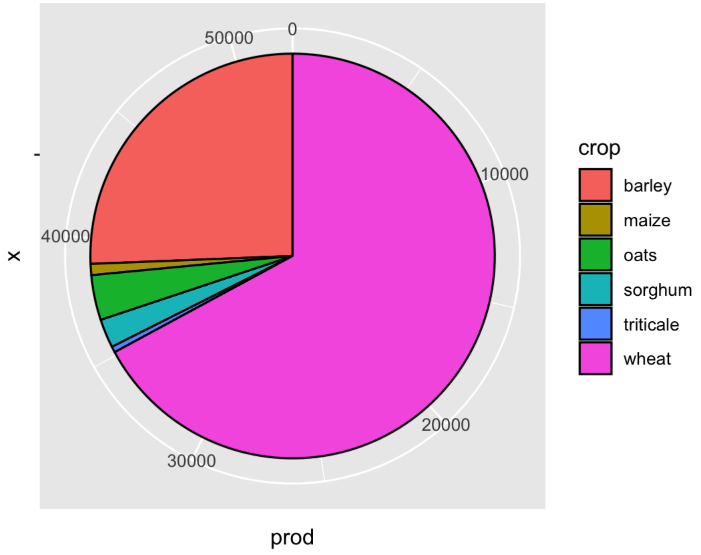
```
df
```

```
##         crop  prod
## 1     barley 13414
## 2    sorghum  1208
## 3      maize   467
## 4       oats  1879
## 5 triticale   247
## 6      wheat 35134
```

```
ggplot(df, aes(x = "", # dummy
               y = prod,
               fill = crop)) +
  geom_col(color = "black")
```



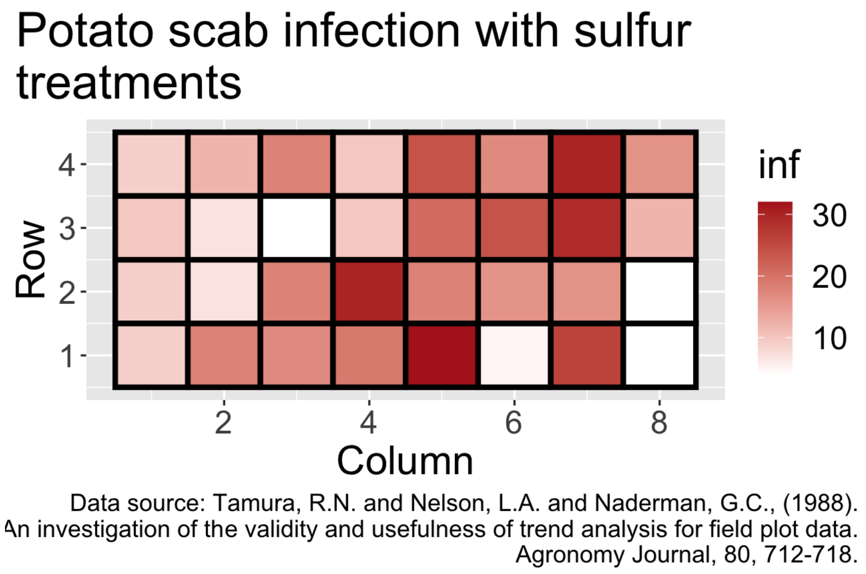
```
ggplot(df, aes(x = "", # dummy
               y = prod,
               fill = crop)) +
  geom_col(color = "black") +
  coord_polar(theta = "y")
```



The difference between a **stacked barplot** and a **pie chart** is that the coordinate system is transformed from the **Cartesian coordinate** to **polar coordinate**.

# Making publication ready plots with ggplot2

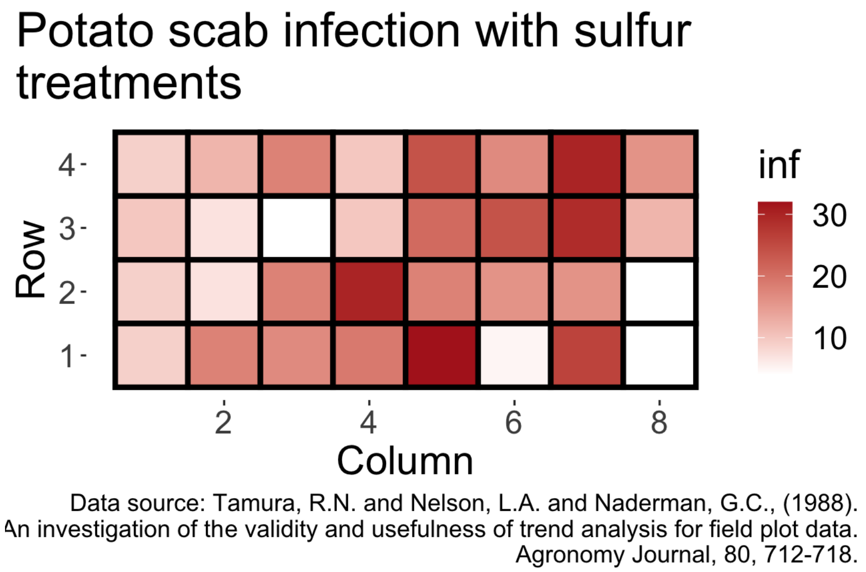
```
ggplot(cochran.crd, aes(col, row, fill = inf)) +
  geom_tile(color = "black", size = 1.3) +
  scale_fill_gradient(low = "white", high = "firebrick") 
  labs(title = "Potato scab infection with sulfur\ntreatm
       x = "Column", y = "Row",
       caption = "Data source: Tamura, R.N. and Nelson, L
  theme(text = element_text(size = 20),
        plot.caption = element_text(size = 12),
        plot.title.position = "plot",
        plot.caption.position = "plot",)
```


Potato scab infection with sulfur treatments

Data source: Tamura, R.N. and Nelson, L.A. and Naderman, G.C., (1988). An investigation of the validity and usefulness of trend analysis for field plot data. Agronomy Journal, 80, 712-718.

# Making publication ready plots with ggplot2

```r
ggplot(cochran.crd, aes(col, row, fill = inf)) +
  geom_tile(color = "black", size = 1.3) +
  scale_fill_gradient(low = "white", high = "firebrick") +
  labs(title = "Potato scab infection with sulfur\ntreatm
       x = "Column", y = "Row",
       caption = "Data source: Tamura, R.N. and Nelson, L
  theme(text = element_text(size = 20),
        plot.caption = element_text(size = 12),
        plot.title.position = "plot",
        plot.caption.position = "plot",
        panel.background = element_blank())
```



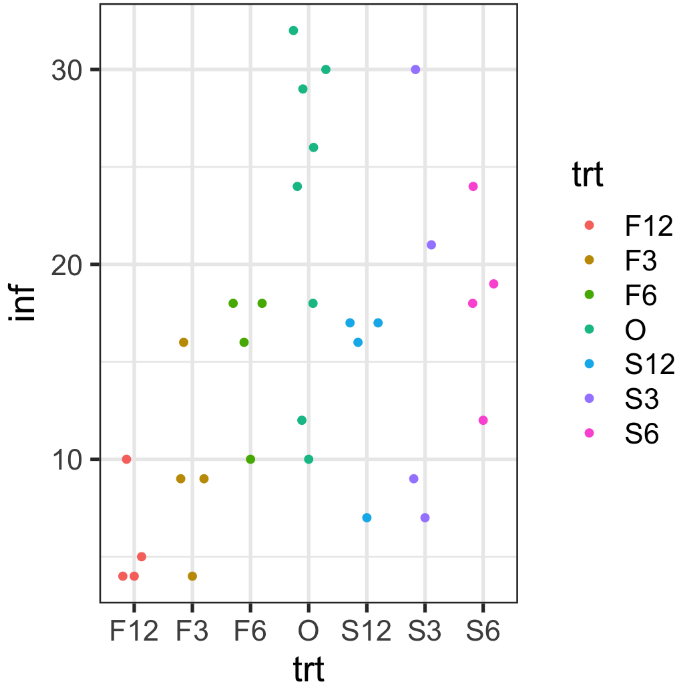
Potato scab infection with sulfur treatments

Data source: Tamura, R.N. and Nelson, L.A. and Naderman, G.C., (1988).
An investigation of the validity and usefulness of trend analysis for field plot data.
Agronomy Journal, 80, 712-718.

# Many extension packages exists for ggplot2

https://exts.ggplot2.tidyverse.org/gallery/

```
ggplot(cochran.crd, aes(trt, inf, color = trt)) +
  ggbeeswarm::geom_quasirandom() +
  theme_bw(base_size = 18)
```



4

# Visualising experimental designs with the deggust R-package

# Visualising experimental designs

> The `deggust` R-package aims to convert `edibble` designs to `ggplot` objects *seamlessly*.

⚠ Currently under developed!

**Package documentation**:

deggust.emitanaka.org

**Source code**:

github.com/emitanaka/deggust

**Name origin**: deggust as in *degust*, and

make **d**esign of **e**xperiments into **`ggplot`** objects

H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016.

# Example: Pig diet experiment

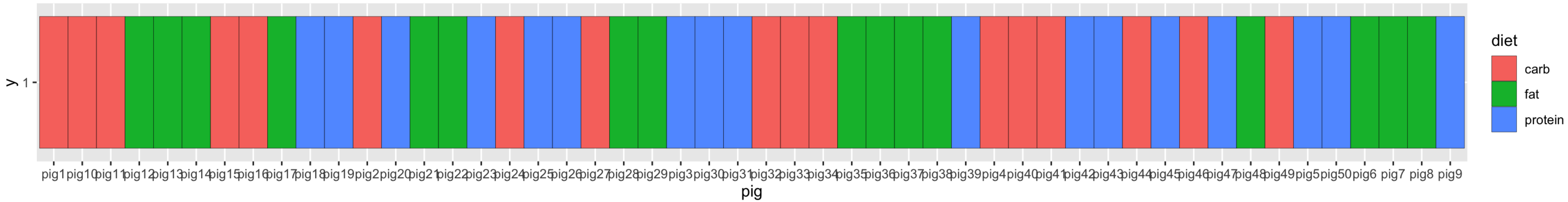
```
library(edibble)
plan <- start_design("Pig diet experiment") %>%
  set_trts(diet = c("carb", "protein", "fat")) %>%
  set_units(pig = 50) %>%
  allot_trts(diet ~ pig) %>%
  assign_trts("random", seed = 1) %>%
  serve_table()

plan

## # An edibble: 50 x 2
##         diet        pig
##     <trt(3)> <unit(50)>
##   1 carb           pig1
##   2 carb           pig2
##   3 protein        pig3
##   4 carb           pig4
##   5 protein        pig5
```

# Visualising designs with ggplot2

```r
library(ggplot2)
plan %>%
  edibble::as_data_frame() %>% # in the future this step will not be needed
  ggplot(aes(pig, "1", fill = diet)) +
  geom_tile(color = "black")
```



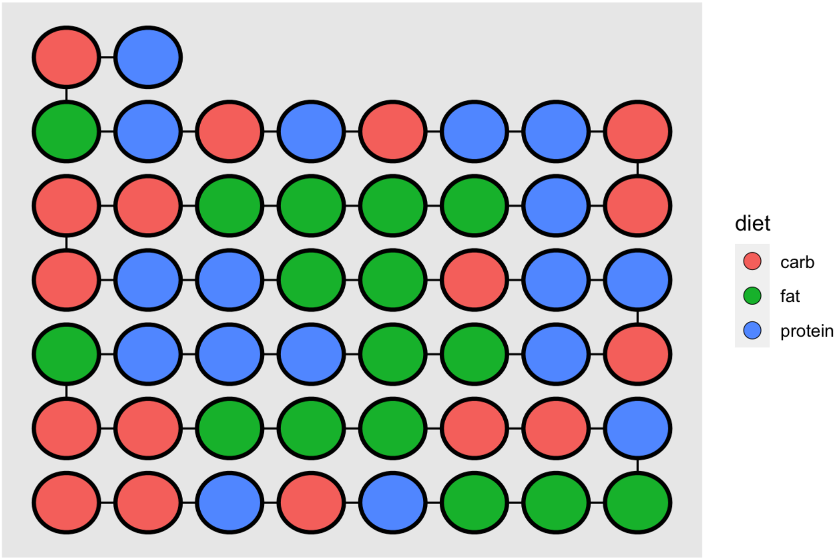
- Slightly painful if you want to *quickly* visualise your design.
- Also not a great visualisation

# deggust::autoplot() Part 1

Just autoplot it!

```
library(deggust)

autoplot(plan)
```



# `deggust::autoplot()` Part 2

- It makes some decision for you of how to plot which can be customised in two ways:

  ❶ modify scale and theme like any ggplot objects!

  ❷ as arguments in the `autoplot` function

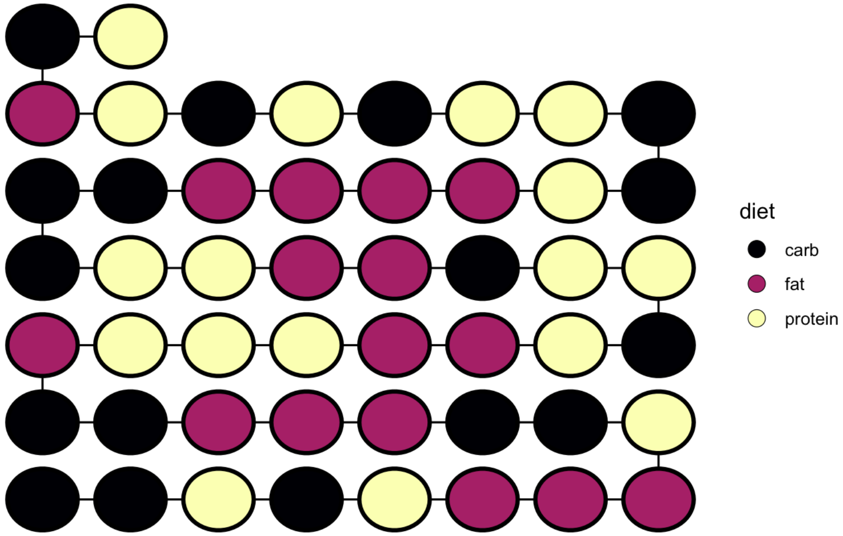
# deggust::autoplot() Part 3

- It makes some decision for you of how to plot which can be customised in two ways:

❶ **modify scale and theme like any ggplot objects!**

❷ as arguments in the `autoplot` function

```
autoplot(plan) +
  # ggplot2 functions below
  theme_void() +
  scale_fill_viridis_d(option = "A")
```



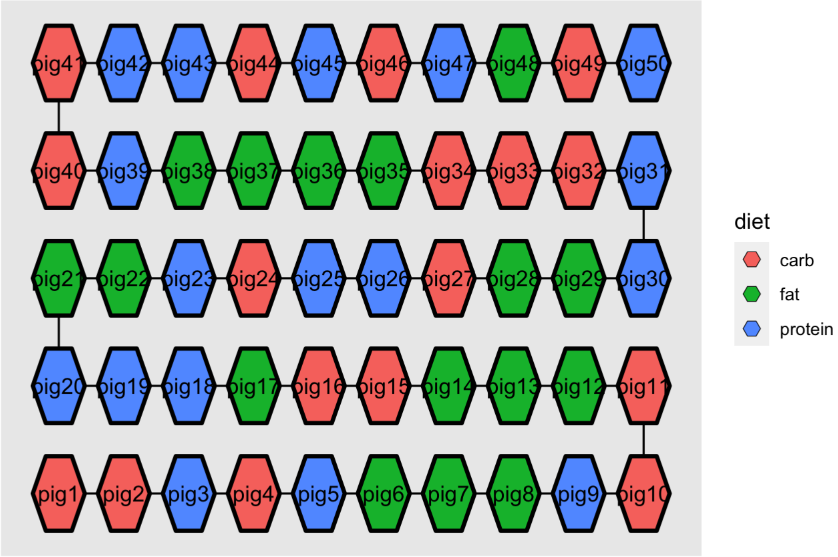
# deggust::autoplot() Part 4

- It makes some decision for you of how to plot which can be customised in two ways:

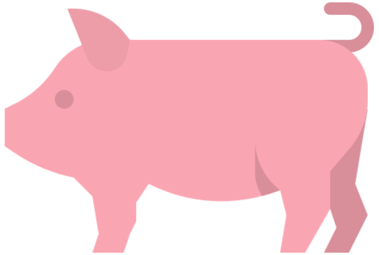
❶ modify scale and theme like any ggplot objects!

❷ as arguments in the `autoplot` function

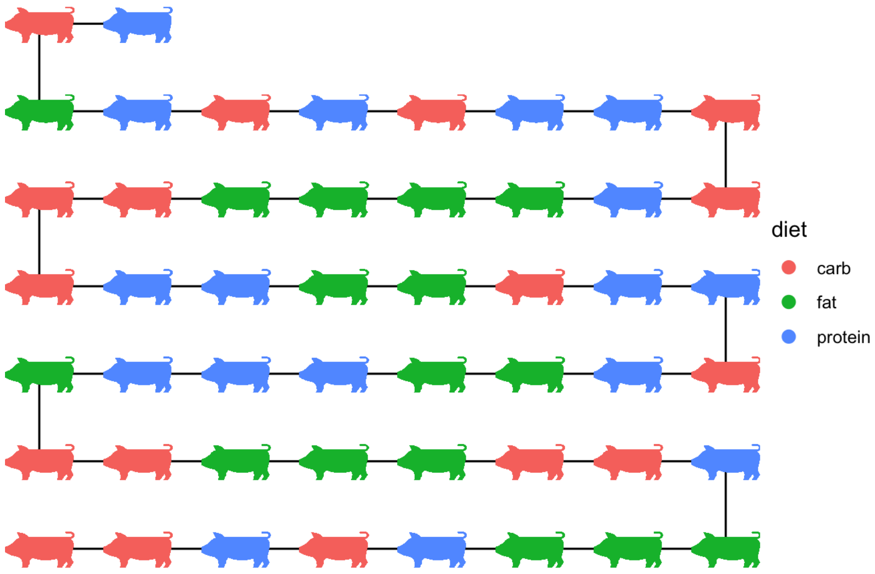
```
autoplot(plan,
         shape = "hexagon",
         text = TRUE,
         aspect_ratio = 0.5)
```



# deggust::autoplot() Part 5



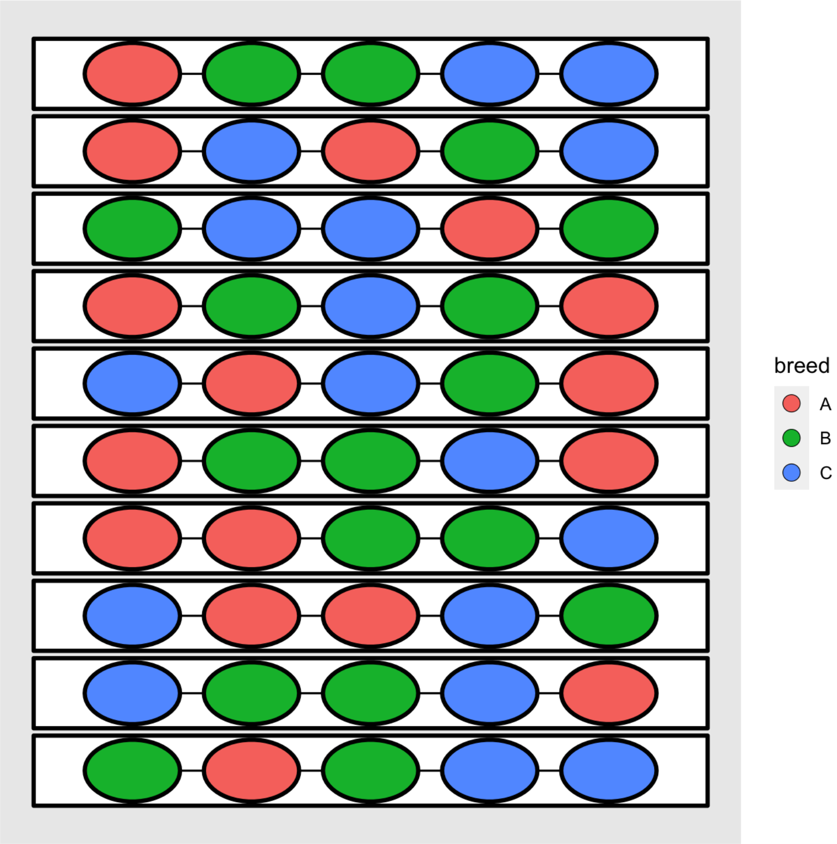
```
autoplot(plan,
         image = "images/pig.png") +
  theme_void()
```



# deggust::autoplot() Part 6

- Nested design

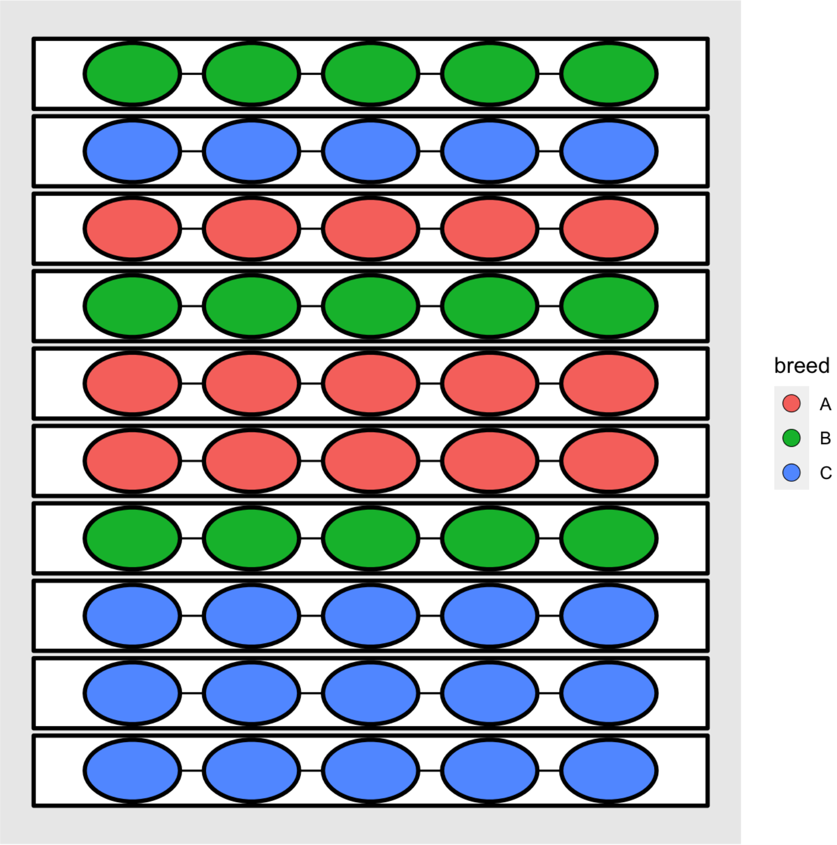
```r
start_design() %>%
  set_units(pen = 10,
            pig = nested_in(pen, 5)) %>%
  set_trts(breed = c("A", "B", "C")) %>%
  allot_trts(breed ~ pig) %>%
  assign_trts("random", seed = 2021) %>%
  serve_table() %>%
  autoplot()
```



# deggust::autoplot() Part 7

- What changed here?

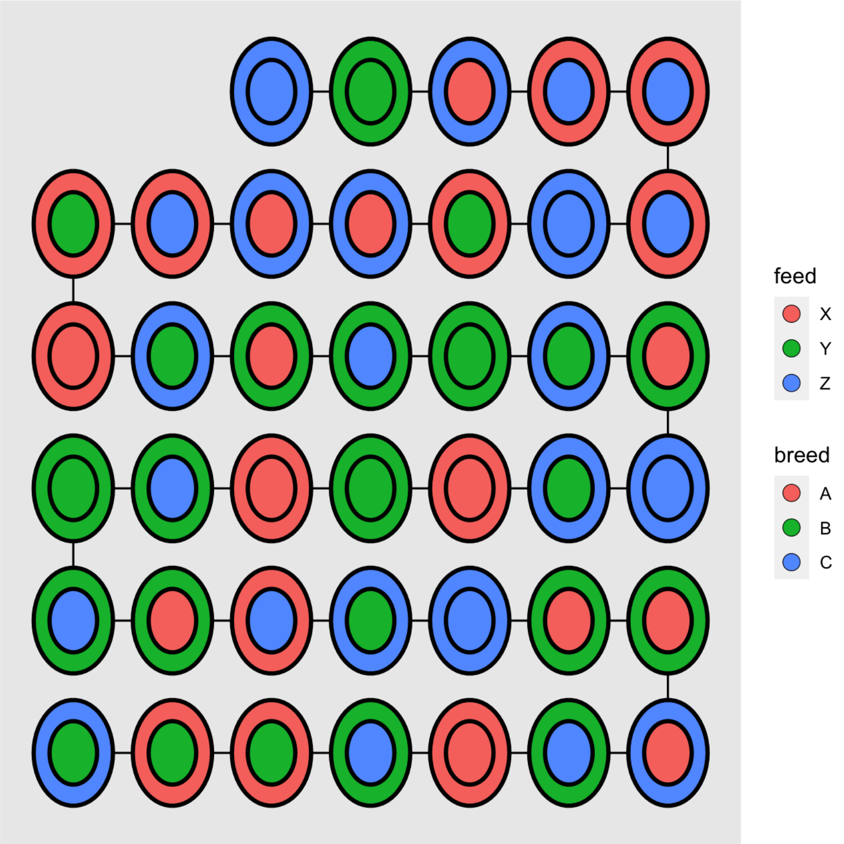
```
start_design() %>%
  set_units(pen = 10,
            pig = nested_in(pen, 5)) %>%
  set_trts(breed = c("A", "B", "C")) %>%
  allot_trts(breed ~ pen) %>%
  assign_trts("random", seed = 2021) %>%
  serve_table() %>%
  autoplot()
```



# deggust::autoplot() Part 8

- Factorial experiment

```
start_design() %>%
  set_units(pig = 40) %>%
  set_trts(breed = c("A", "B", "C"),
           feed = c("X", "Y", "Z")) %>%
  allot_trts(breed:feed ~ pig) %>%
  assign_trts("random", seed = 2021) %>%
  serve_table() %>%
  autoplot()
```

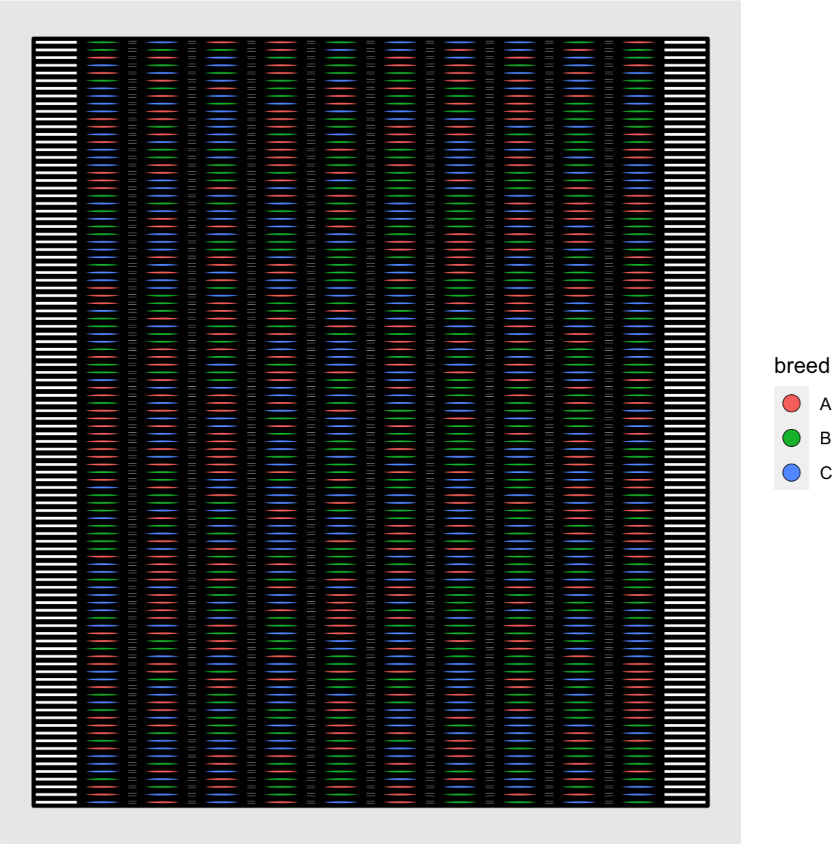


- Note: scale will be fixed so it's easier to distinguish between different treatment factors

# deggust::autoplot() Part 10

- Is your design too big to fit in the plot?

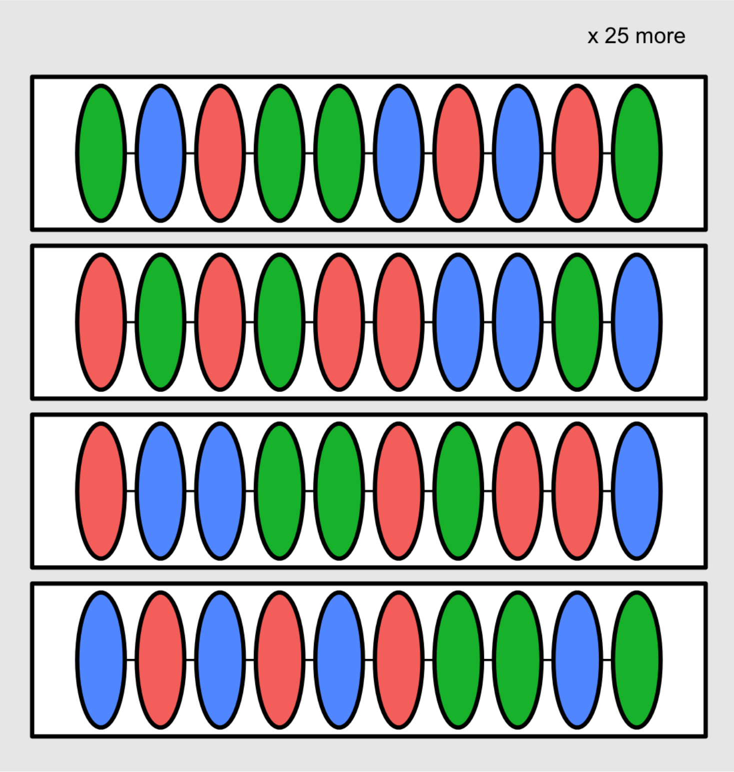
```
start_design() %>%
  set_units(pen = 100,
            pig = nested_in(pen, 10)) %>%
  set_trts(breed = c("A", "B", "C")) %>%
  allot_trts(breed ~ pig) %>%
  assign_trts("random", seed = 2021) %>%
  serve_table() %>%
  autoplot()
```



# deggust::autoplot() Part 10

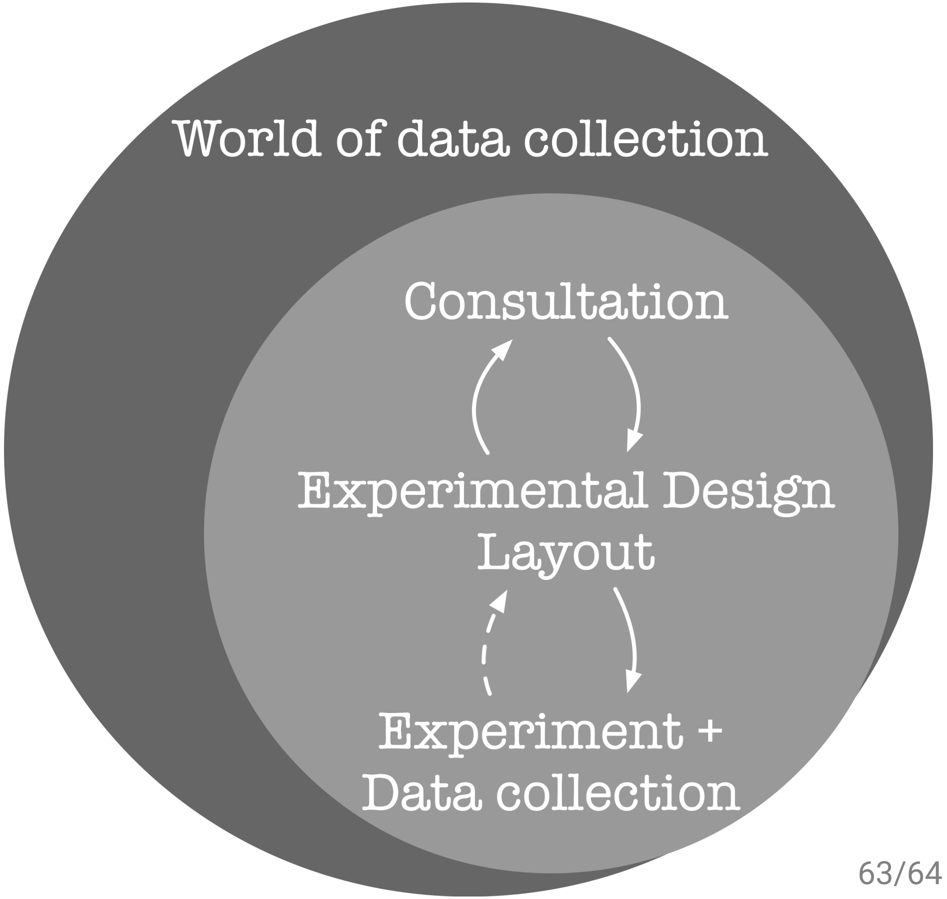
- Is your design too big to fit in the plot?
- Subset it!

```
start_design() %>%
  set_units(pen = 100,
            pig = nested_in(pen, 10)) %>%
  set_trts(breed = c("A", "B", "C")) %>%
  allot_trts(breed ~ pig) %>%
  assign_trts("random", seed = 2021) %>%
  serve_table() %>%
  dplyr::filter(pen %in% c("pen1", "pen2", "pen3", "pen4
  autoplot() +
  annotate("text", x = 10, y = 4.7, label = "x 25 more")
```



# Summary

- The grammar of experimental designs is an abstract computational framework that encourages a higher-order thinking by enforcing the experimental structure and context
- `edibble` is designed to be user friendly and accommodate natural order of thinking for specifying experimental structure
- The grammar makes each step modular... you can easily extend it (like `deggust`) or mix-and-match methods
- This makes it easier to leverage existing functionalities in `edibble` so other developers can focus on what they want to do the most
- And hopefully this framework becomes a common base that promotes collaboration and knowledge sharing



# Thanks for listening!

- Slides: emitanaka.org/slides/stats4bio2021/deggust
- `edibble` package documentation: edibble.emitanaka.org
- `edibble` source code: github.com/emitanaka/edibble
- `deggust` package documentation: deggust.emitanaka.org
- `deggust` source code: github.com/emitanaka/deggust
- emi.tanaka@monash.edu @statsgen

- Feature requests or issues with `edibble` or `deggust`? Submit or upvote here: github.com/emitanaka/edibble/issues, github.com/emitanaka/deggust/issues, send me an email or tell me now!