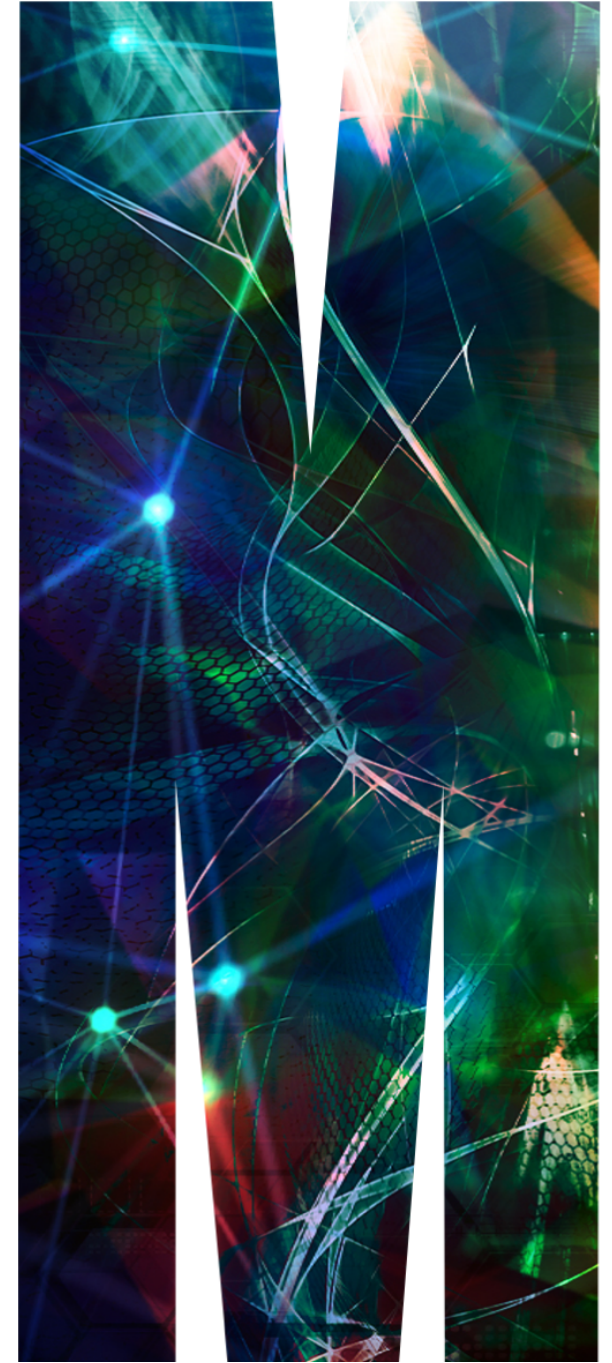# Visual Inference and Experimental Design

Presenter: *Emi Tanaka*

Department of Econometrics and Business Statistics

✉ emi.tanaka@monash.edu   🐦 @statsgen

📅 Wed 22nd Sep 2021

Emi Tanaka

Lecturer in statistics

Monash University

✉ emi.tanaka@monash.edu

🐦 @statsgen

💻 emitanaka

🌐 emitanaka.org

📖 Table of Contents

Introduction to Visual Inference

- Model diganostics are cruicial if inferences are to be made on the model

- A common model diagnostic is to look at residual plots and to search for any patterns

- If there is a pattern, it is indicative of a misspecified model

- **Do you see a pattern on the plot in the right?**

Residual plot

- In statistics, we are careful in assessing **numerical statistics** by taking into account its uncertainty

- BUT we are *quite informal in assessing plots*
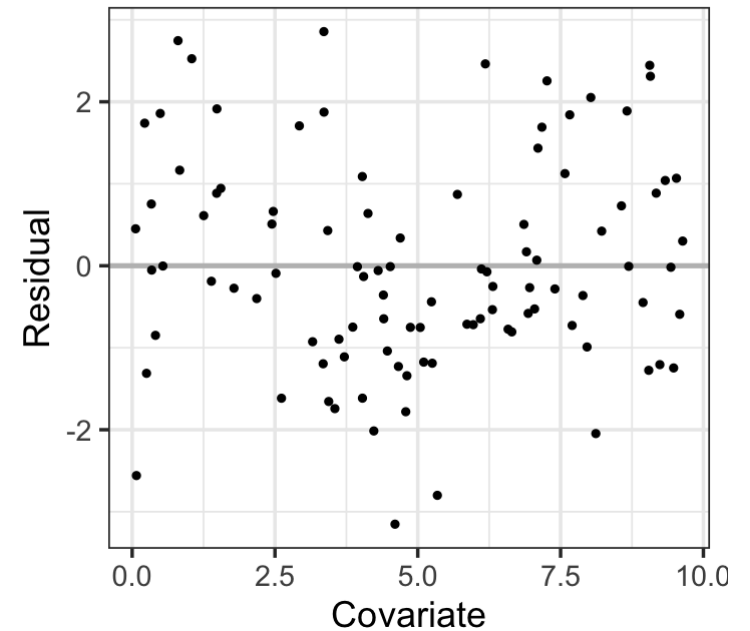
- Data plots are also statistics

- ***Data plots ought to be treated with the same rigour as numerical statistics***
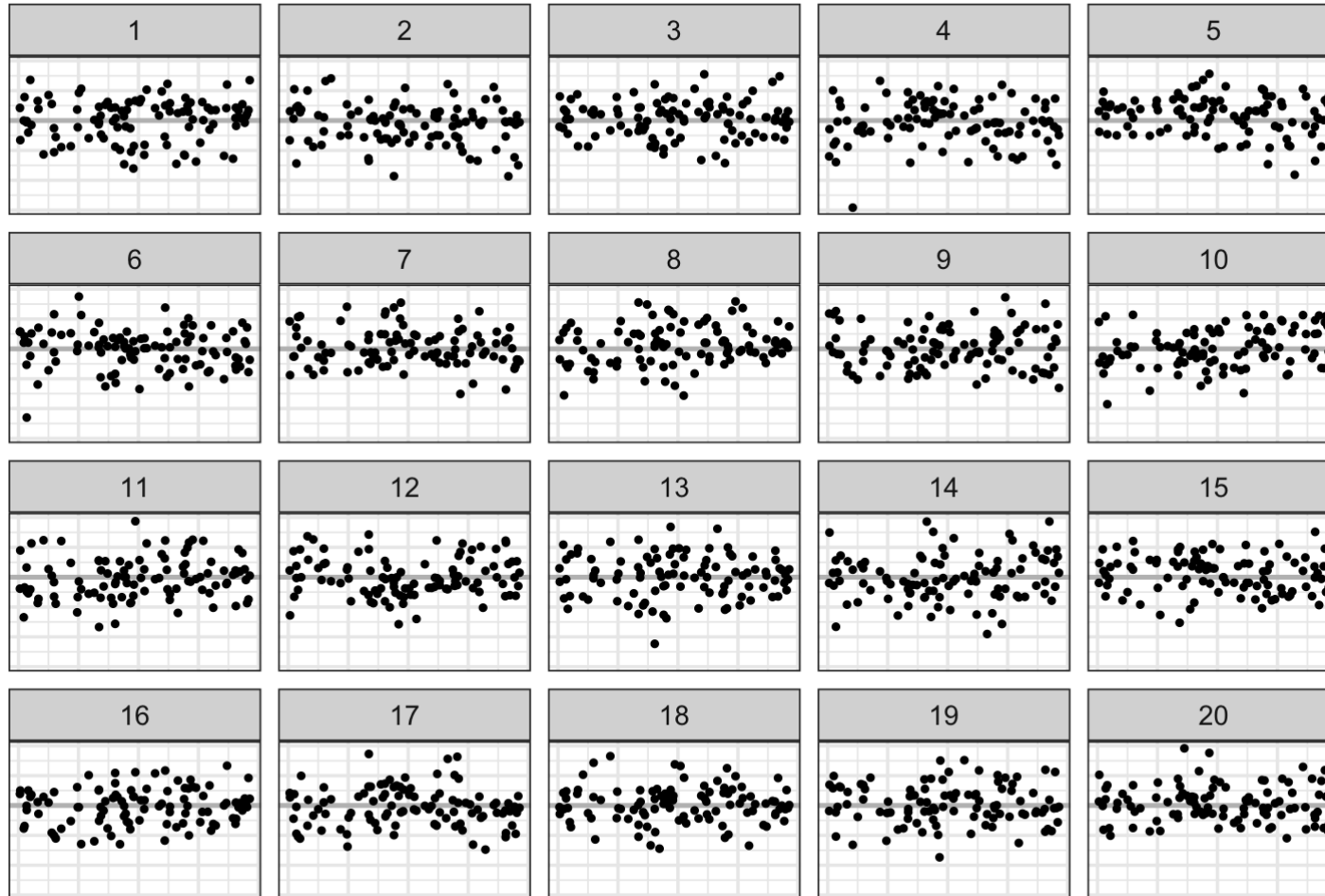
Frequentist framework

Hypotheses

↓

Assumptions

↓

Test Statistic

↓

P-value

↓

Conclusion
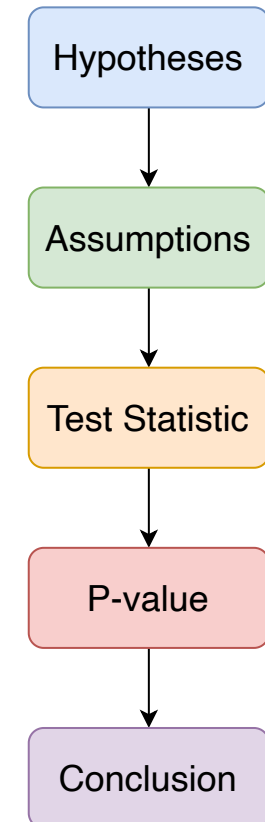
- **Hypotheses**: $H_0 : e_i \sim NID(0, \sigma^2)$ vs. $H_1 :$ not H_0

- **Assumptions**: σ^2 is known and is equal to maximum likelihood estimate

- **Test statistic**: Residual plot

- **P-value**: ???

- We can use the approach by Buja et al. (2009):

  1. Strip away context from data plot so there is no bias

  2. Generate the null data under H_0

  3. Create $k - 1$ "null plots" by using visual encoding choice as data plot but using the null data

  4. Embed the data plot in a random position within the lineup of null plots

  5. Ask m observers which plot is the most different in the lineup and let X be the # of observers who detected the data plot

  6. Assuming observers are independent and equal visual ability, $X \sim B(m, 1/k)$.

  7. And the P-value (or "see-value") is given as $P(X \geq x)$.

Buja, Andreas, Dianne Cook, Heike Hofmann, Michael Lawrence, Eun-Kyung Lee, Deborah F. Swayne, and Hadley Wickham. 2009. "Statistical Inference for Exploratory Data Analysis and Model Diagnostics." Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences 367 (1906): 4361–83.

Which plot looks the most different to you?

- The data generating process is

$$y = 1 + 2x + sin(x) + e$$

where $e \sim N(0, 1)$

- The fitted model is
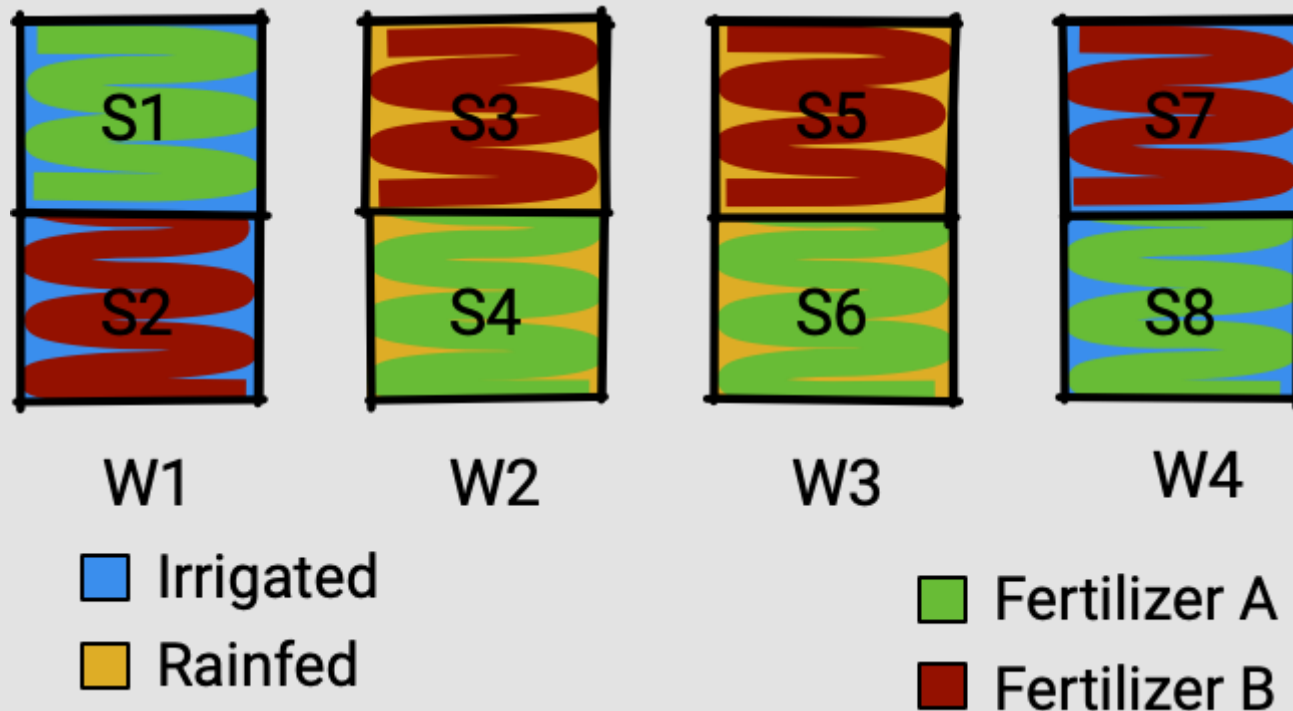
$$y = \beta_0 + \beta_1 x + e$$

where $e \sim N(0, \sigma^2)$

- The position of the data plot is 12

The Grammar of
Experimental Designs

CONTEXT — Study of **two irrigation methods** and **two fertilizer brands** on the yields of a crop.

W1 W2 W3 W4

- ☐ (blue) Irrigated
- ☐ (yellow) Rainfed
- ☐ (green) Fertilizer A
- ☐ (dark red) Fertilizer B

The *grammar of experimental design* with `edibble`

```r
library(edibble)
start_design("split-plot")
```

MAIN POINTS

- Initialises an object with a special class
- It doesn't really contain much at this stage

OUTPUT

```
## split-plot
```

The *grammar of experimental design* with `edibble`

```r
library(edibble)
start_design("split-plot") %>%
  set_units(wholeplot = 4)
```

MAIN POINTS

- Now we specify that there are 4 wholeplots
- The object holds the *intermediate construct* of an experimental design

OUTPUT

```
## split-plot
## └──wholeplot (4 levels)
```

The *grammar of experimental design* with `edibble`

```r
library(edibble)
start_design("split-plot") %>%
  set_units(wholeplot = 4) %>%
  set_units(subplot = nested_in(wholeplot, 2))
```

MAIN POINTS

- Then we specify that there are 2 subplots for each wholeplot

- The name of the units are not restricted to "wholeplot" and "subplot"; the user can use *any* name

OUTPUT

```
## split-plot
## └─wholeplot (4 levels)
##     └─subplot (8 levels)
```

The *grammar of experimental design* with `edibble`

```r
library(edibble)
start_design("split-plot") %>%
  set_units(wholeplot = 4,
            subplot = nested_in(wholeplot, 2))
```

MAIN POINTS

- We can combine the "set_units"

OUTPUT

```
## split-plot
## └─wholeplot (4 levels)
##    └─subplot (8 levels)
```

The *grammar of experimental design* with `edibble`

```r
library(edibble)
start_design("split-plot") %>%
  set_units(wholeplot = 4,
            subplot = nested_in(wholeplot, 2)) %>%
  set_trts(water = c("irrigated", "rainfed"),
           fertilizer = c("A", "B"))
```

- We now set the treatments
- There are 2 treatment factors (water and fertilizer) with 2 levels each

```
## split-plot
## ├──wholeplot (4 levels)
## |   └──subplot (8 levels)
## ├──water (2 levels)
## └──fertilizer (2 levels)
```

The *grammar of experimental design* with `edibble`

```r
library(edibble)
start_design("split-plot") %>%
  set_trts(water = c("irrigated", "rainfed"),
           fertilizer = c("A", "B")) %>%
  set_units(wholeplot = 4,
            subplot = nested_in(wholeplot, 2))
```

MAIN POINTS

- You can set the treatment before the units

OUTPUT

```
## split-plot
## ├──water (2 levels)
## ├──fertilizer (2 levels)
## └──wholeplot (4 levels)
##    └──subplot (8 levels)
```

The *grammar of experimental design* with `edibble`

```r
library(edibble)
start_design("split-plot") %>%
  set_trts(water = c("irrigated", "rainfed")) %>%
  set_units(wholeplot = 4) %>%
  set_trts(fertilizer = c("A", "B")) %>%
  set_units(subplot = nested_in(wholeplot, 2))
```

- Or mix it as you like (although subplot needs to appear after wholeplot)
- The edibble system tries to support a natural order to define the experimental structure

```
## split-plot
## ├──water (2 levels)
## ├──wholeplot (4 levels)
## |   └──subplot (8 levels)
## └──fertilizer (2 levels)
```

The *grammar of experimental design* with `edibble`

```r
library(edibble)
start_design("split-plot") %>%
  set_units(wholeplot = 4,
            subplot = nested_in(wholeplot, 2)) %>%
  set_trts(water = c("irrigated", "rainfed"),
           fertilizer = c("A", "B")) %>%
  allocate_trts(water ~ wholeplot,
                fertilizer ~ subplot)
```

MAIN POINTS

- We then define the mapping of treatment to units
- The print output doesn't look any different

OUTPUT

```
## split-plot
## ├──wholeplot (4 levels)
## │   └──subplot (8 levels)
## ├──water (2 levels)
## └──fertilizer (2 levels)
```

The *grammar of experimental design* with `edibble`

```r
library(edibble)
start_design("split-plot") %>%
  set_units(wholeplot = 4,
            subplot = nested_in(wholeplot, 2)) %>%
  set_trts(water = c("irrigated", "rainfed"),
           fertilizer = c("A", "B")) %>%
  allocate_trts(water ~ wholeplot,
                fertilizer ~ subplot) %>%
  randomise_trts()
```

MAIN POINTS

- Then we randomise the treatment to units
- Again, the output doesn't look different

OUTPUT

```
## split-plot
## ├──wholeplot (4 levels)
## │   └──subplot (8 levels)
## ├──water (2 levels)
## └──fertilizer (2 levels)
```

The *grammar of experimental design* with `edibble`

```r
library(edibble)
start_design("split-plot") %>%
  set_units(wholeplot = 4,
            subplot = nested_in(wholeplot, 2)) %>%
  set_trts(water = c("irrigated", "rainfed"),
           fertilizer = c("A", "B")) %>%
  allocate_trts(water ~ wholeplot,
                fertilizer ~ subplot) %>%
  randomise_trts() %>%
  serve_table()
```

MAIN POINTS

- Finally, we signal that we're done constructing the design
- The output gets converted to a data frame

OUTPUT

```
## # An edibble: 8 x 4
##     wholeplot    subplot      water f
##     <unit(4)> <unit(8)>  <trt(2)>
## 1 wholeplot1   subplot1 rainfed
## 2 wholeplot1   subplot2 rainfed
## 3 wholeplot2   subplot3 irrigated
## 4 wholeplot2   subplot4 irrigated
## 5 wholeplot3   subplot5 irrigated
## 6 wholeplot3   subplot6 irrigated
## 7 wholeplot4   subplot7 rainfed
## 8 wholeplot4   subplot8 rainfed
```

The *grammar of experimental design* with `edibble`

```r
library(edibble)
start_design("Modified allocation") %>%
  set_units(wholeplot = 4,
            subplot = nested_in(wholeplot, 2)) %>%
  set_trts(water = c("irrigated", "rainfed"),
           fertilizer = c("A", "B")) %>%
  allocate_trts(water:fertilizer ~ subplot) %>%
  randomise_trts() %>%
  serve_table()
```

MAIN POINTS

- Let's say we modify the treatment allocation
- The resulting design is what we call "randomised complete block design"

OUTPUT

```
## # An edibble: 8 x 4
##     wholeplot    subplot      water f
##     <unit(4)> <unit(8)>  <trt(2)>
## 1 wholeplot1  subplot1 irrigated
## 2 wholeplot1  subplot2 irrigated
## 3 wholeplot2  subplot3 rainfed
## 4 wholeplot2  subplot4 rainfed
## 5 wholeplot3  subplot5 rainfed
## 6 wholeplot3  subplot6 rainfed
## 7 wholeplot4  subplot7 irrigated
## 8 wholeplot4  subplot8 irrigated
```

Main contributions

- The *grammar of experimental design* is a (programming language agnostic) framework that functionally maps the fundamental components of the experiment to an object oriented programming system to build and modify an experimental design

- The *edibble* R-package is an implementation of the grammar of experimental design in the R language
  </> https://github.com/emitanaka/edibble

- The approach is designed to be *human-friendly* and accommodate natural order of thinking for specifying experimental structure

- The approach also *promotes higher order thinking about experimental design*, e.g. the difference between a split-plot design and a randomised complete block design is pronounced in code

- Finally, the grammar makes each step modular... you can *easily extend* or *mix-and-match methods*

Summary

- **Adopting good computational practices** is important to me for transparency and reproducibility (e.g. version control, sharing code, open-source tools)

- **Visual inference** extends statistical inference to data visualisation

- I proactively work in **software development with human-centered design for the design of experiments**

- You can find these slides and *code* at:

  🔗 emitanaka.org/slides-DARE-LOOP
  🐙 https://github.com/emitanaka/slides-DARE-LOOP