ANOVA continued, and scrollytelling

Overview

Review and continuation of ANOVAs

- Unbalanced data
- Repeated measures/block designs and random effects models

Joining data frames

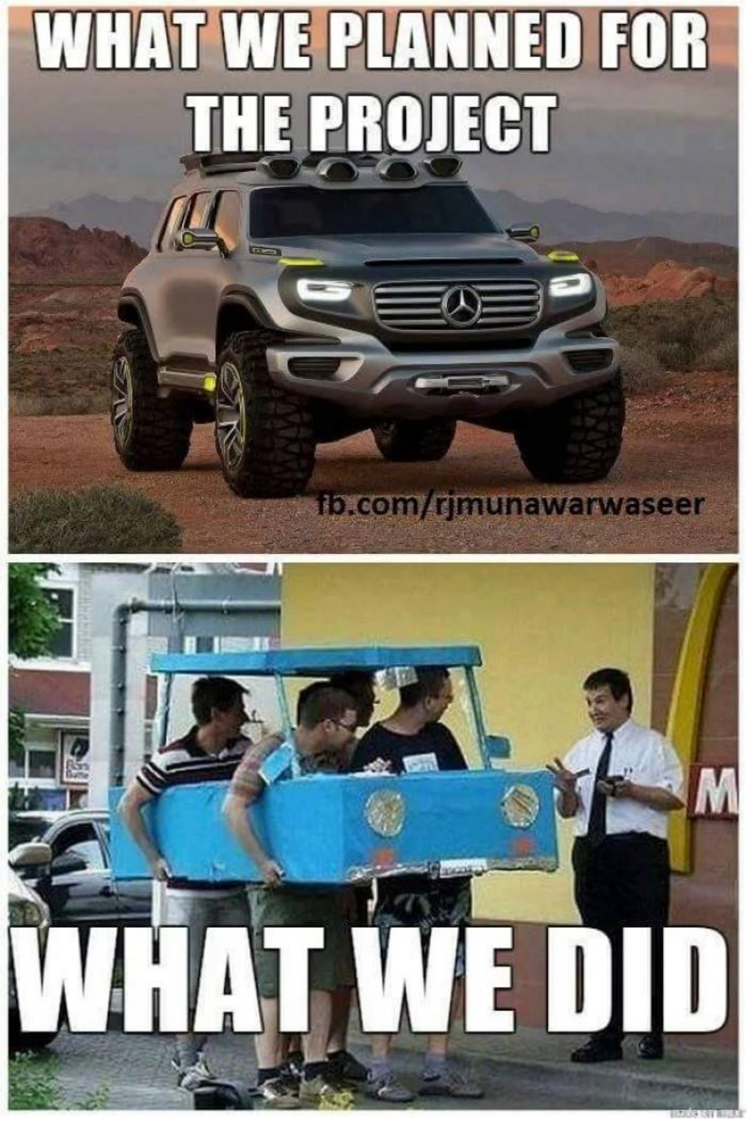
Creating scrollytelling documents with Closeread

How are final projects going?

Any questions about the final project?

I will have extra office hours this Friday from 1:30-3pm

Final exam review session on Tuesday December 10th at 4pm



ANOVA review

An Analysis of Variance (ANOVA) can be viewed as:

- A hypothesis test comparing multiple means
- A model for predicting means from categorical variables

In a **factorial ANOVA**, we model the response variable y as a function of **more than one** categorical predictor

For a two-way ANOVA we have:

$$y_{ijk} = \mu + \alpha_j + \beta_k + \gamma_{jk} + \varepsilon_{ijk}$$

Random error for the ijkth data point

ith response when:
- factor A has level j
- Factor B has level k

Overall mean

Main effect for factor A at level j

Main effect for factor B at level k

Specific interaction for jth level of A and kth level of B

Two-way ANOVA hypotheses

Main effect for A

H_0: $\alpha_1 = \alpha_2 = \ldots = \alpha_{J-1} = 0$

H_A: $\alpha_j \neq 0$ for some j

Main effect for B

H_0: $\beta_1 = \beta_2 = \ldots = \beta_{K-1} = 0$

H_A: $\beta_k \neq 0$ for some k

Interaction effect:

H_0: All $\gamma_{jk} = 0$

H_A: $\gamma_{jk} \neq 0$ for some j, k

Where:

α_j: is the “effect” for factor A at level j

β_k: is the “effect” for factor B at level k

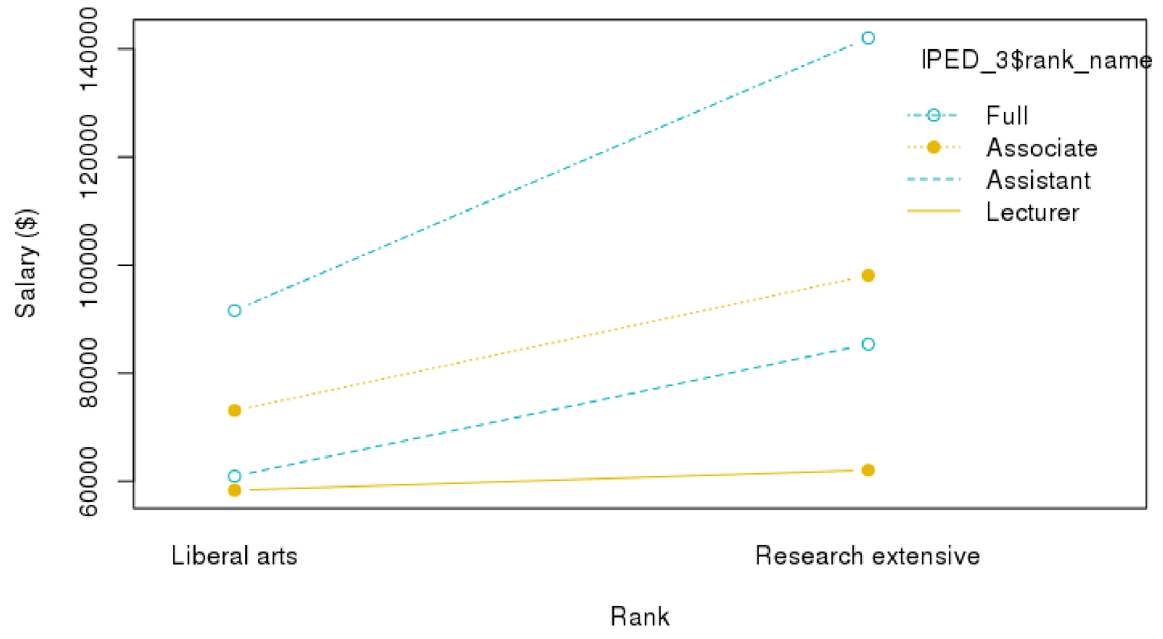
γ_{jk} : is the interaction between level j of factor A, and level k of factor B.

$$y_{ijk} = \mu + \alpha_j + \beta_k + \gamma_{jk} + \varepsilon_{ijk}$$

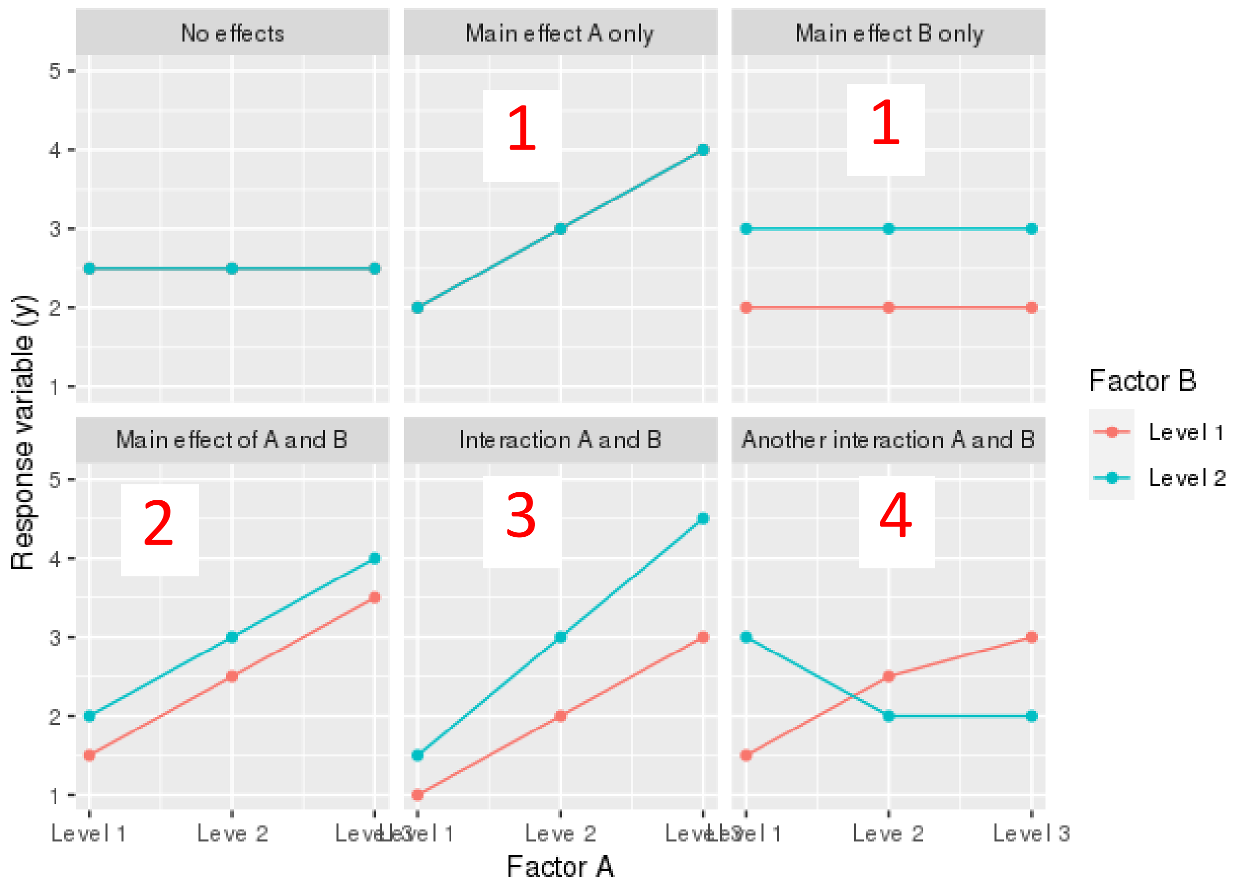
Interaction plots

Interaction plots can help us visualize main effects and interactions

- Plot the levels of one of the factors on the x-axis
- Plot the levels of the other factor as separate lines



Interpreting interaction through interaction plots



What are examples we have seen in class of the interactions in plots 1, 2, 3 and 4?

Complete and balanced designs

Complete factorial design: at least one measurement for each possible combination of factor levels

- E.g., in a two-way ANOVA for factors A and B, if there are K levels for factor A, and J levels for factor B, then there needs to be at least one measurement for each of the KJ levels

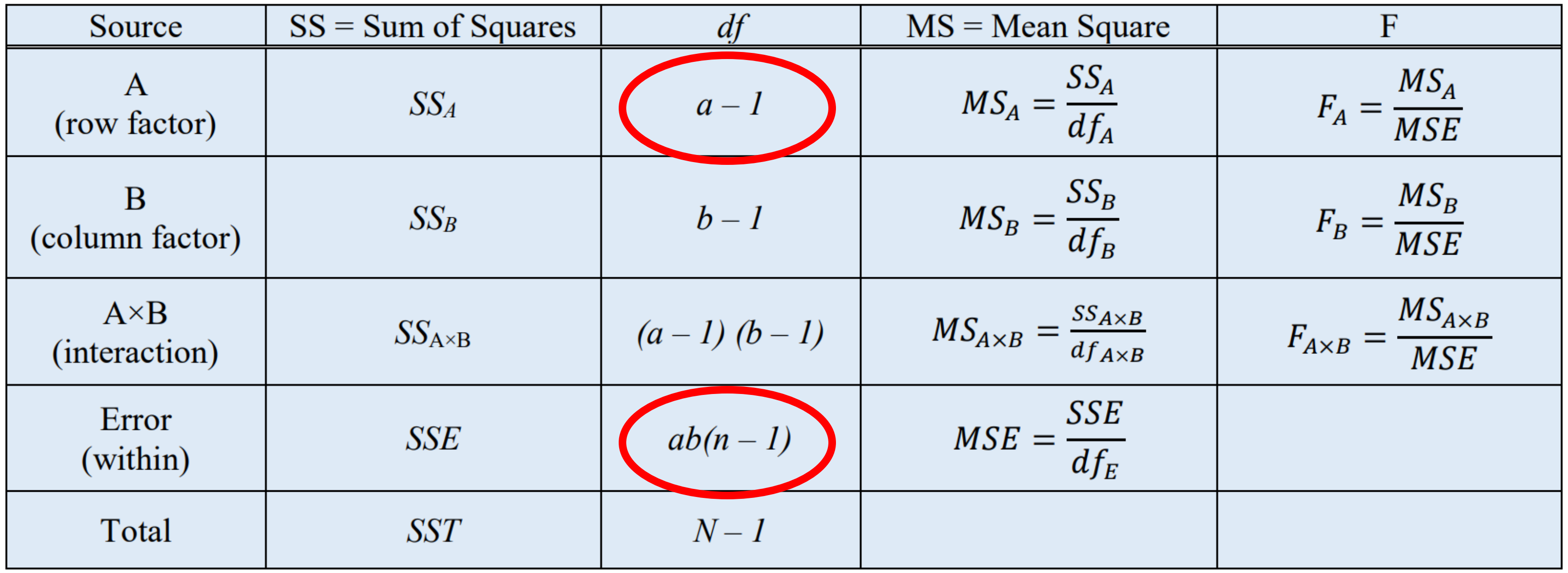
Balanced design: the sample size is the same for all combination of factor levels

- E.g., there are the same number of samples in each of the KJ level combinations.
- The computations and interpretations for non-balanced designs are a bit harder.

Unbalanced designs

We can get p-values by from an F-distribution with the appropriate degrees of freedom

Two-way ANOVA table with interaction



Source	SS = Sum of Squares	df	MS = Mean Square	F
A (row factor)	SS_A	$a-1$	$MS_A = \frac{SS_A}{df_A}$	$F_A = \frac{MS_A}{MSE}$
B (column factor)	SS_B	$b-1$	$MS_B = \frac{SS_B}{df_B}$	$F_B = \frac{MS_B}{MSE}$
A×B (interaction)	$SS_{A\times B}$	$(a-1)(b-1)$	$MS_{A\times B} = \frac{SS_{A\times B}}{df_{A\times B}}$	$F_{A\times B} = \frac{MS_{A\times B}}{MSE}$
Error (within)	SSE	$ab(n-1)$	$MSE = \frac{SSE}{df_E}$	
Total	SST	$N-1$		

```
df1 <- a - 1
df2 <- ab(n-1)
pf(F_stat,  df1, df2,
            lower.tail = FALSE)
```

where "a" is the number of levels for factor A, etc.

For unbalanced designs, <u>there are different ways </u>to compute the sum of squares, and hence one can get different p-values

- The problem is analogous to multicollinearity. If two explanatory variables are correlated either can account for the variability in the response data

Unbalanced designs

Type I sum of squares: (also called sequential sum of squares) the order that terms are entered in the model matters

- SS(A) is taken into account before SS(B) is considered etc.
- anova(lm(y ~ A*B)) gives different results than using anova(lm(y ~ B*A))

Type II and Type III sum of squares: the order that that terms are entered into the model does not matter.

- For each factor, SS(A), SS(B), SS(AB) is taken into account after all other factors are added
- Car::Anova(lm(y ~ A*B) , type = "III") is the same as car::Anova(lm(y ~ B*A) , type = "III")

Repeated measures ANOVA

In a **repeated measures ANOVA**, the same case/observational units are measured at each factor level

Example: Do people prefer chocolate, butterscotch or caramel sauce?

Between subjects experiment: different people rate chocolate, butterscotch, and caramel sauce

- Run a between subjects ANOVA (as we have done before)

Within subjects experiment: each person in the experiment gives ratings for all three toppings

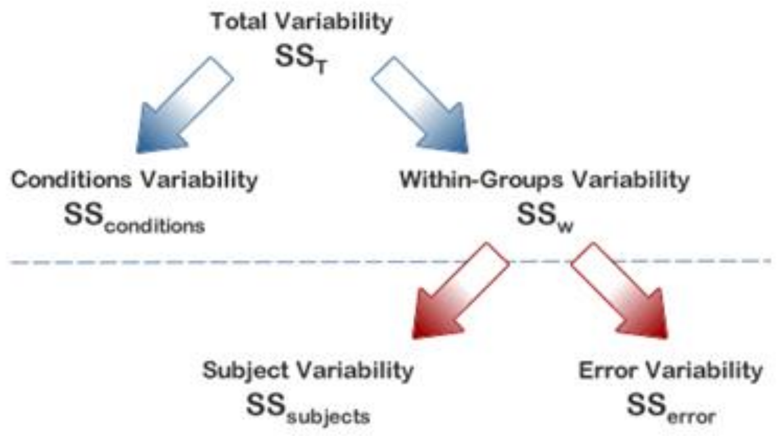
- Run a repeated measures ANOVA

Repeated measures ANOVA

$$F = \frac{\text{between-group variability}}{\text{within-group variability}} = \frac{\frac{1}{K-1}\sum_{i=1}^{K} n_i(\bar{y}_i - \bar{y}_{tot})^2}{\frac{1}{N-K}\sum_{i=1}^{K}\sum_{j=1}^{n_i}(y_{ij} - \bar{y}_i)^2}$$

The advantages of a repeated measures ANOVA is that we can potentially reduce a lot of the variability between the cases

- This is a generalization of a paired t-test to more than two population means



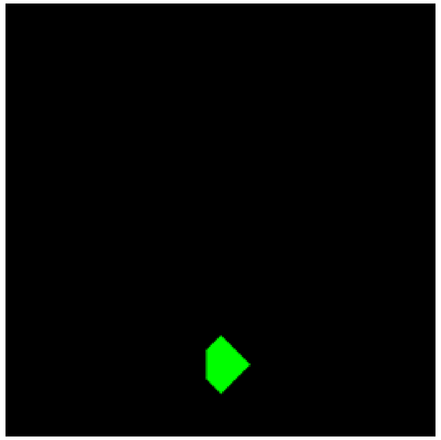
To run a repeated measures ANOVA, we use a factor called ID (or participant, etc.) that has a unique value for each observational unit

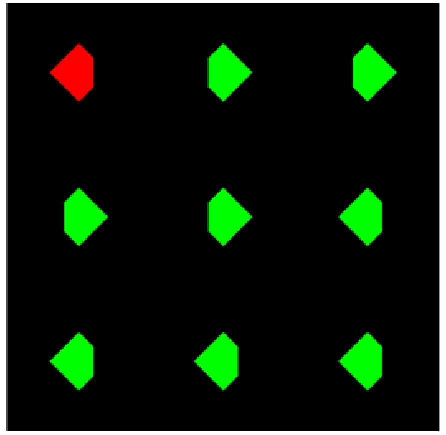
```
aov(reaction_time ~ condition * position +  participant,
    data = popout_log_data)
```

Homework 10: The ANOVA model – popout data

Participants engaged in a reaction time task where they needed to respond as *quickly* and as *accurately* as possible

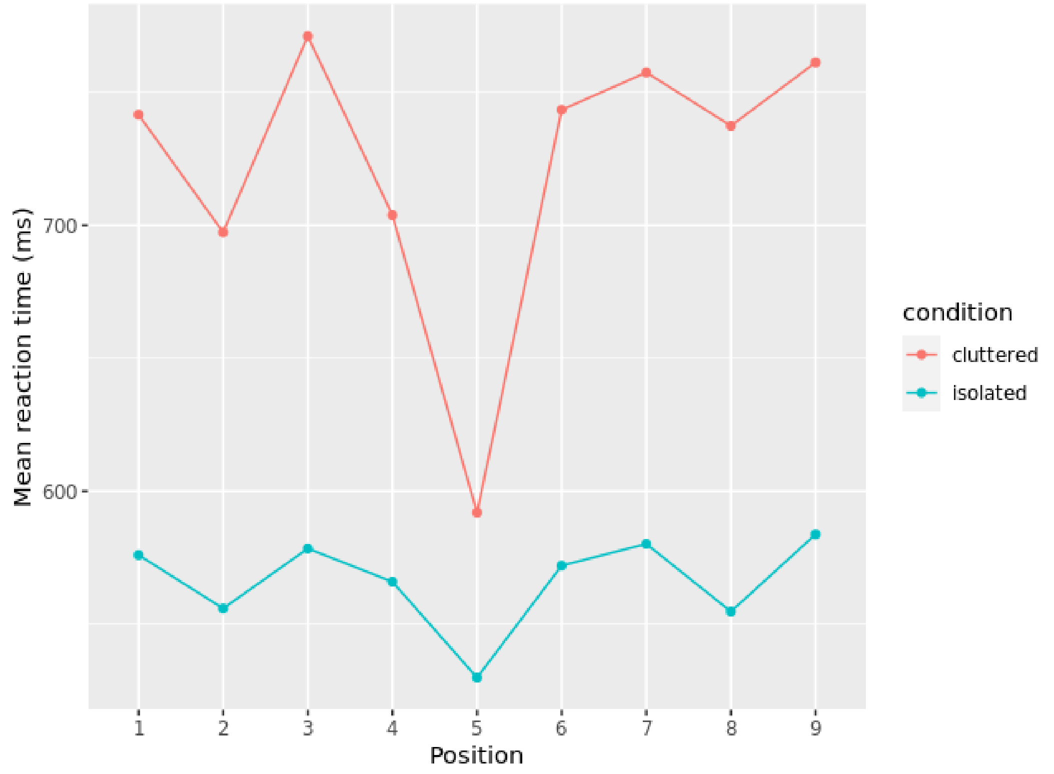
The experiment had a 9 x 2 x 2 x 2 ***factorial*** design

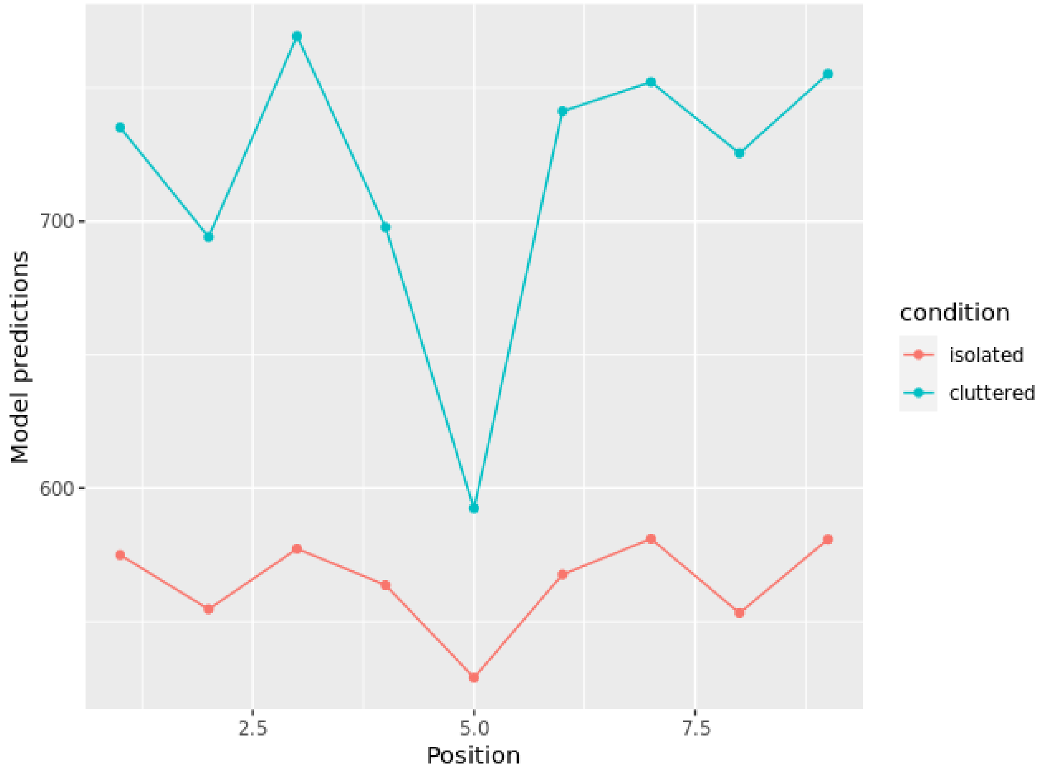




```
aov(reaction_time ~ condition + position + color + cut_dir + correct + block_number + participant +
    same_position_as_last + same_color_as_last + position * condition, data = popout_data)
```

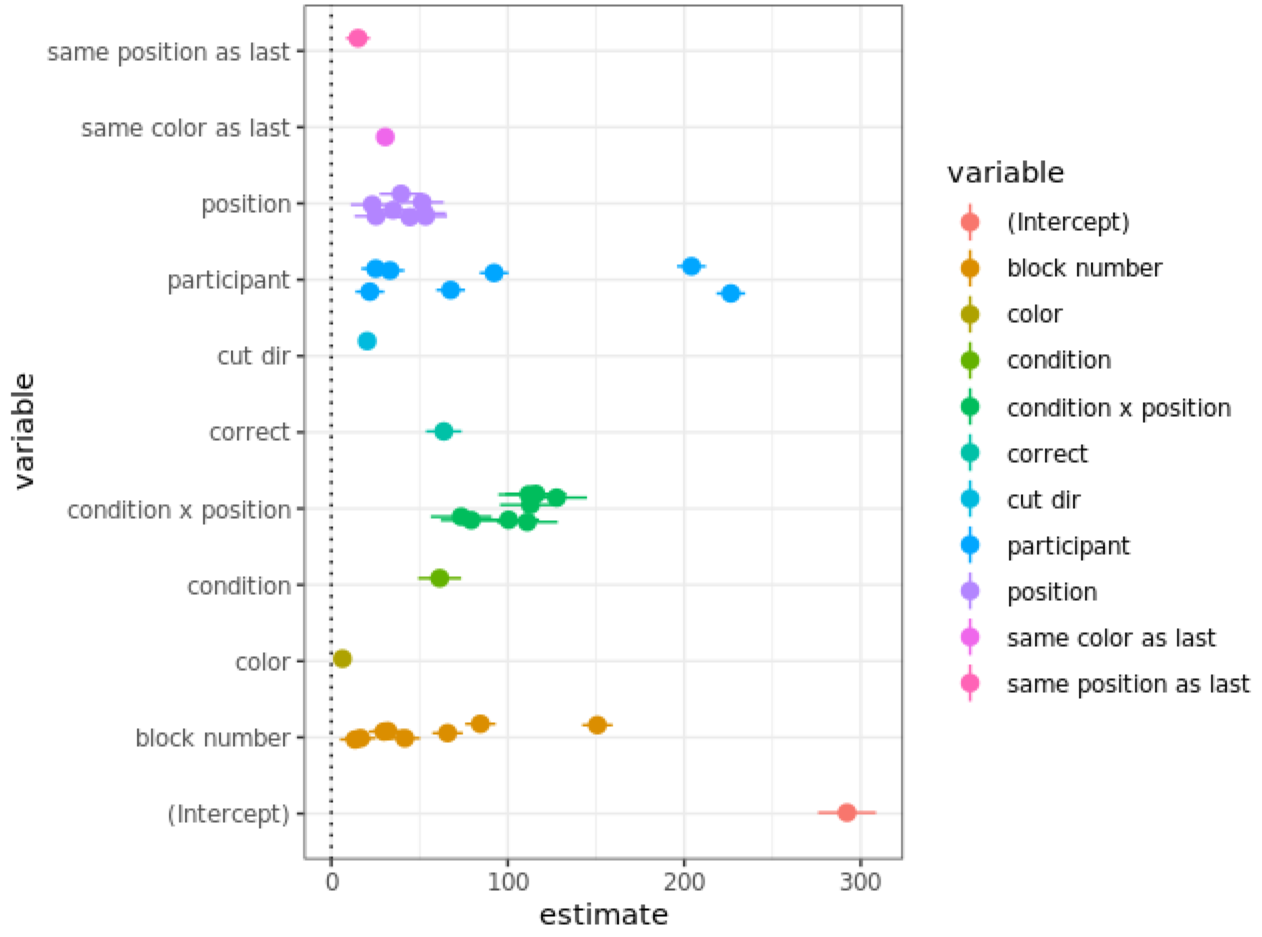
The ANOVA model – popout data

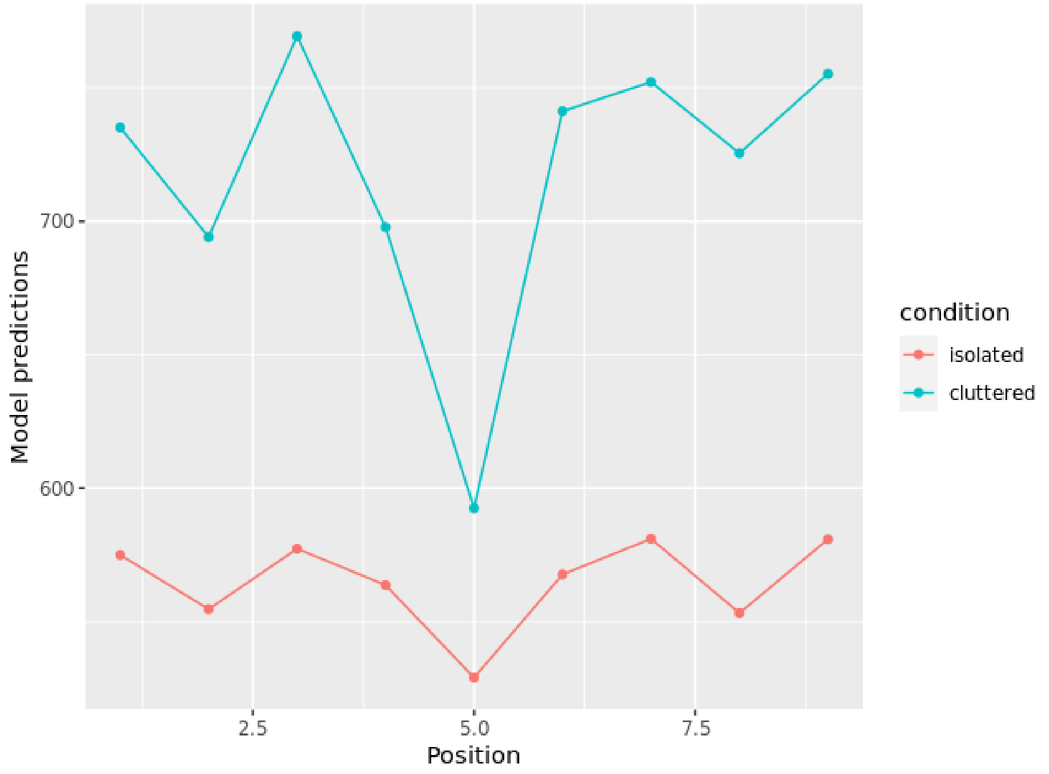




```
aov(reaction_time ~ condition + position + color + cut_dir + correct + block_number + participant +
same_position_as_last + same_color_as_last + position * condition, data = popout_data)
```

The ANOVA model – popout data





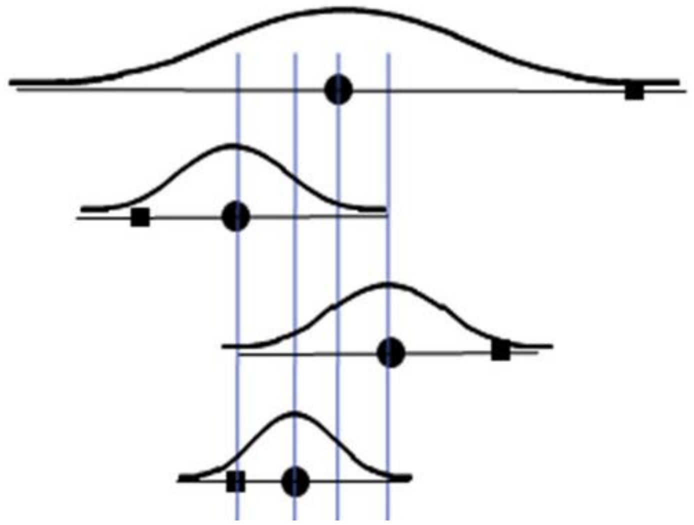
aov(reaction_time ~ condition + position + color + cut_dir + correct + block_number + participant + same_position_as_last + same_color_as_last + position * condition, data = popout_data)

Brief mention: random effects models

In a random effects ANOVA, factor levels are viewed as being randomly generated from an underlying distribution, rather than having a fixed number of levels

For example, we could view participants in an experiment as being a random sample from participants in a population.

- Rather a separately ID for each participant, we just estimate a mean and standard deviation for the underlying population
- This leads to few parameters and hence more degrees of freedom



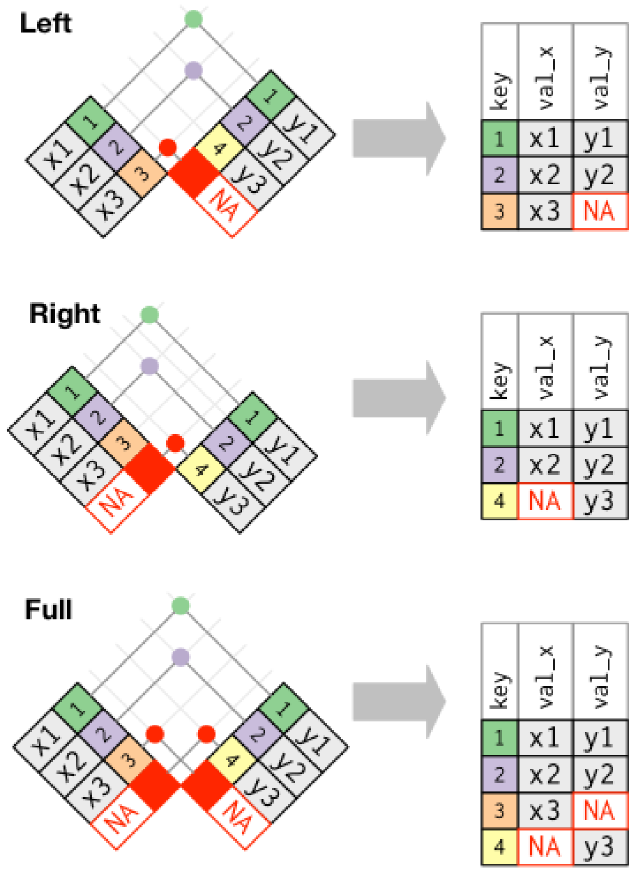
You can run mixed effects models in R using the lme4 package

- This is beyond what we will do in this class :/

Let’s explore these topics in R…

Continuation of tidyverse packages useful for your projects

Joining data frames



Left

key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Right

key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

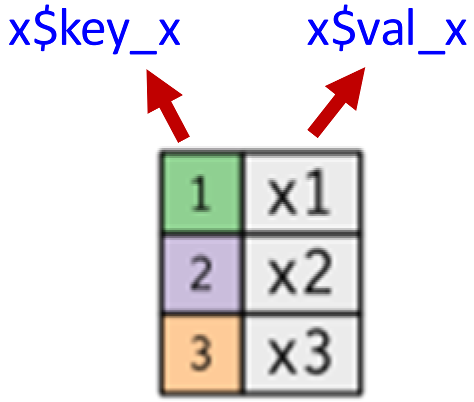
Full

key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

Left and right tables

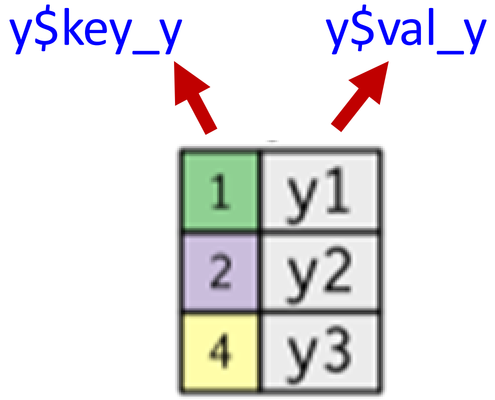
Suppose we have two data frames called x and y

- x have two variables called key_x, and val_x
- y has two variables called key_y and val_y


x$key_x
x$val_x

1	x1
2	x2
3	x3

Data frame x


y$key_y
y$val_y

1	y1
2	y2
4	y3

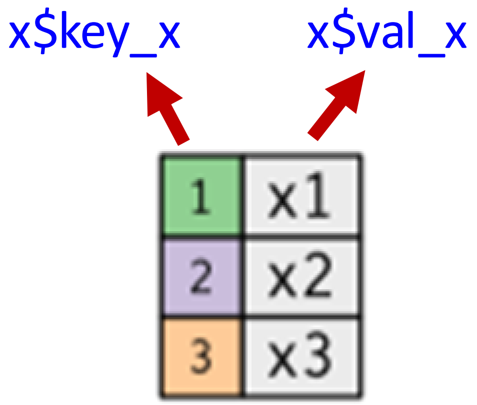
Data frame y

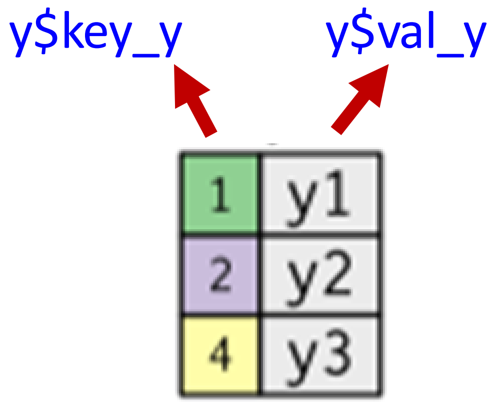
SDS230:download_data('x_y_join.rda')

Left and right tables

Suppose we have two data frames called x and y

- x have two variables called key_x, and val_x
- y has two variables called key_y and val_y





Data frame x **Data frame y**

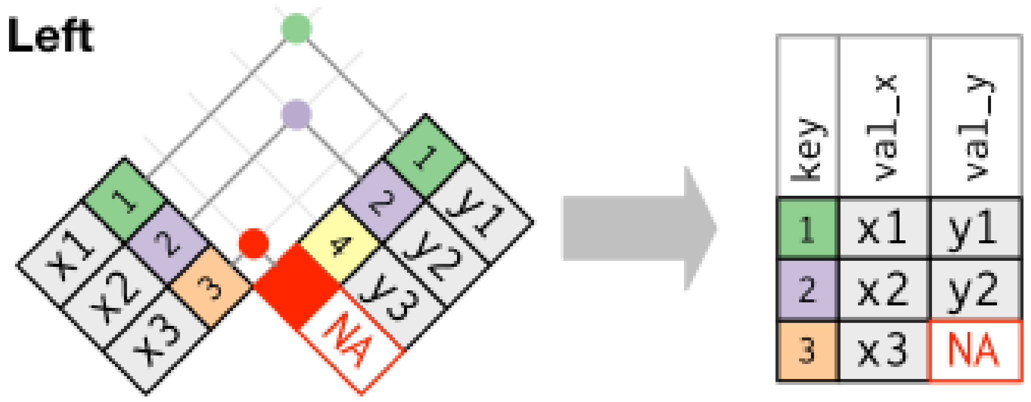
Joins have the general form:

```
join(x, y, by = c("key_x" = "key_y"))
```

Left joins

Left joins keep all rows in the left table.

Data from right table is added when there is a matching key, otherwise NA as added.



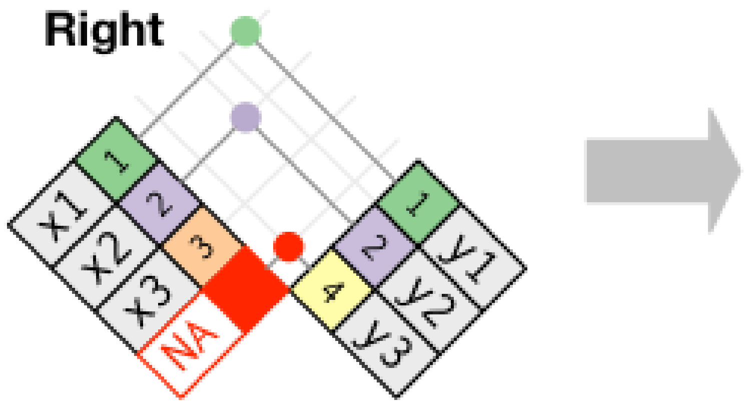
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

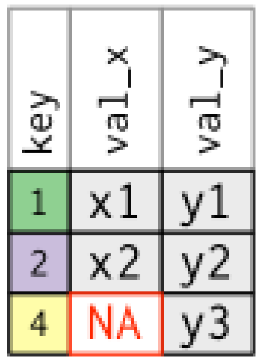
```
> left_join(x, y, by = c("key_x" = "key_y"))
```

Right joins

Right joins keep all rows in the right table.

Data from left table added when there is a matching key, otherwise NA as added.



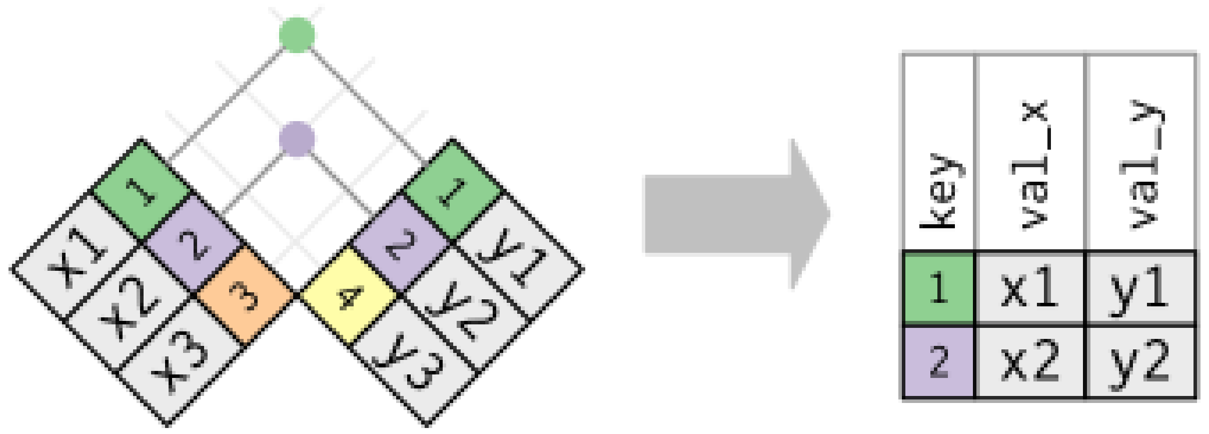


key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

```
> right_join(x, y, by = c("key_x" = "key_y"))
```

Inner joins

Inner joins only keep rows in which there are matches between the keys in both tables.



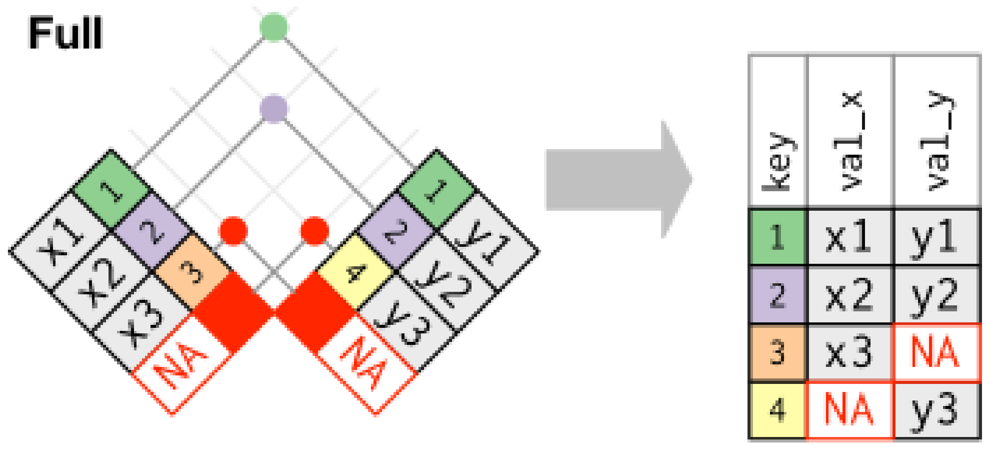
key	val_x	val_y
1	x1	y1
2	x2	y2

```
> inner_join(x, y, by = c("key_x" = "key_y"))
```

Full joins

Full joins keep all rows in both table.

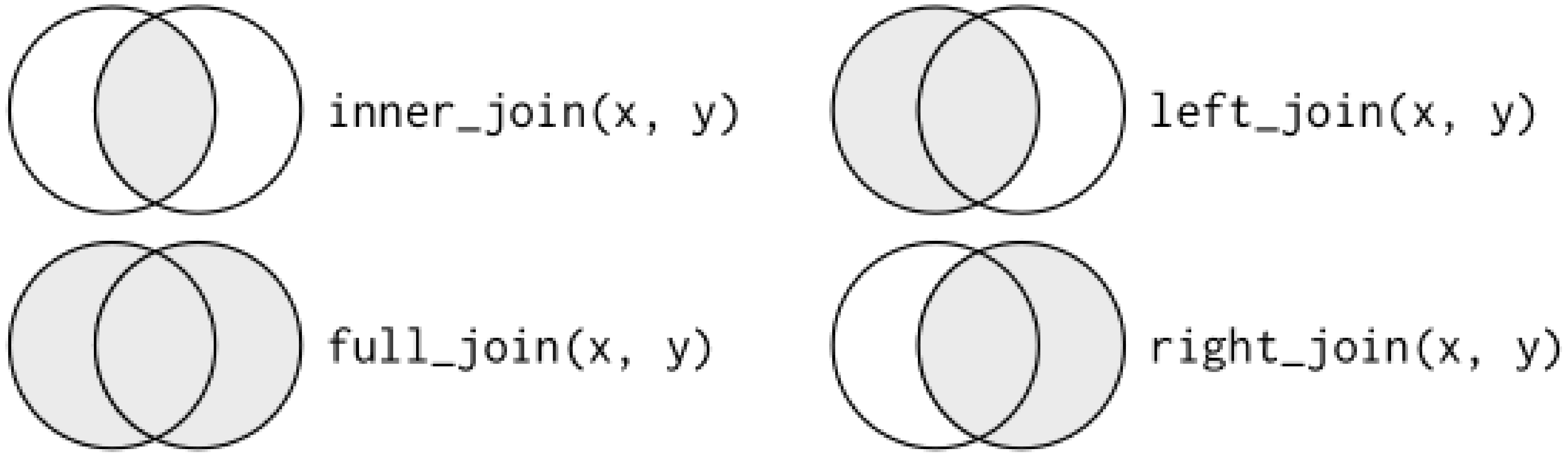
NAs are added where there are no matches.



key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

```
> full_join(x, y, by = c("key_x" = "key_y"))
```

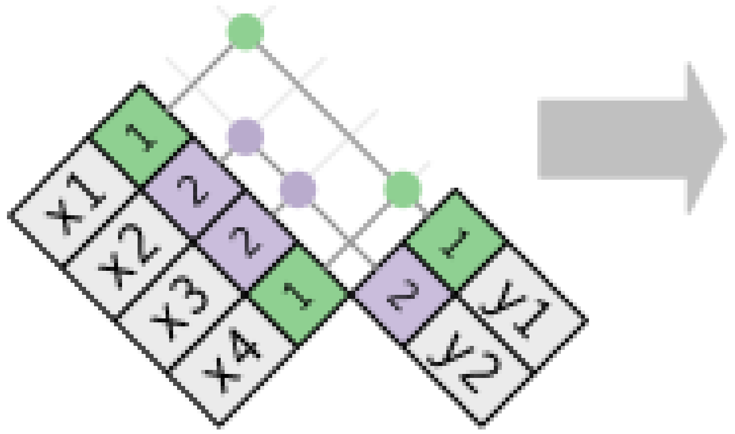
Summary

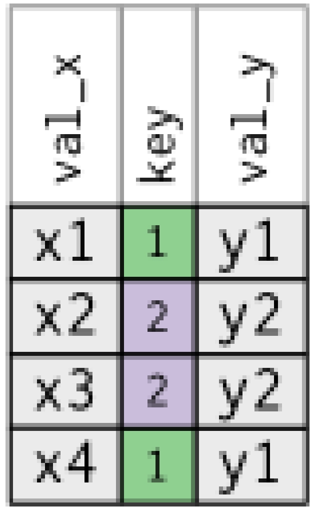


Duplicate keys

Duplicate keys are useful if there is a many-to-one relationship

- e.g., duplicates are useful in the left table when doing a left join



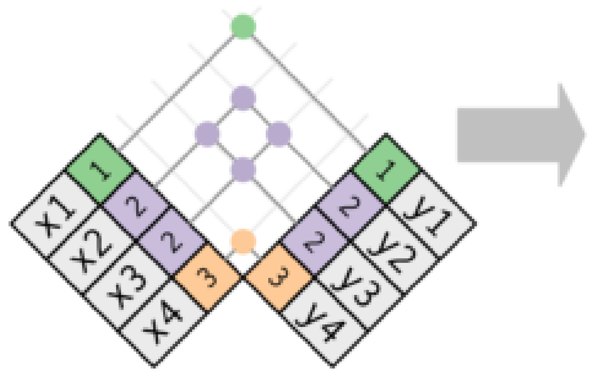


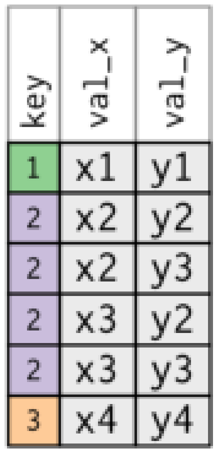
val_x	key	val_y
x1	1	y1
x2	2	y2
x3	2	y2
x4	1	y1

Duplicate keys

If both tables have duplicate keys you get all possible combinations of joined values (Cartesian product).

- **This is usually an error!**





key	val_x	val_y
1	x1	y1
2	x2	y2
2	x2	y3
2	x3	y2
2	x3	y3
3	x4	y4

Always check the output size after you join a table because even if there is not a syntax error you might not get the table you are expecting!

- You can check how many rows a data frame has using the nrow() function

Duplicate keys

To deal with duplicate keys in both tables, we can join the tables using multiple keys in order to make sure that each row is uniquely specified.

We can do this using the syntax:

```
join(x2, y2, by = c("key1_x" = "key1_y", "key2_x" = "key2_y"))
```

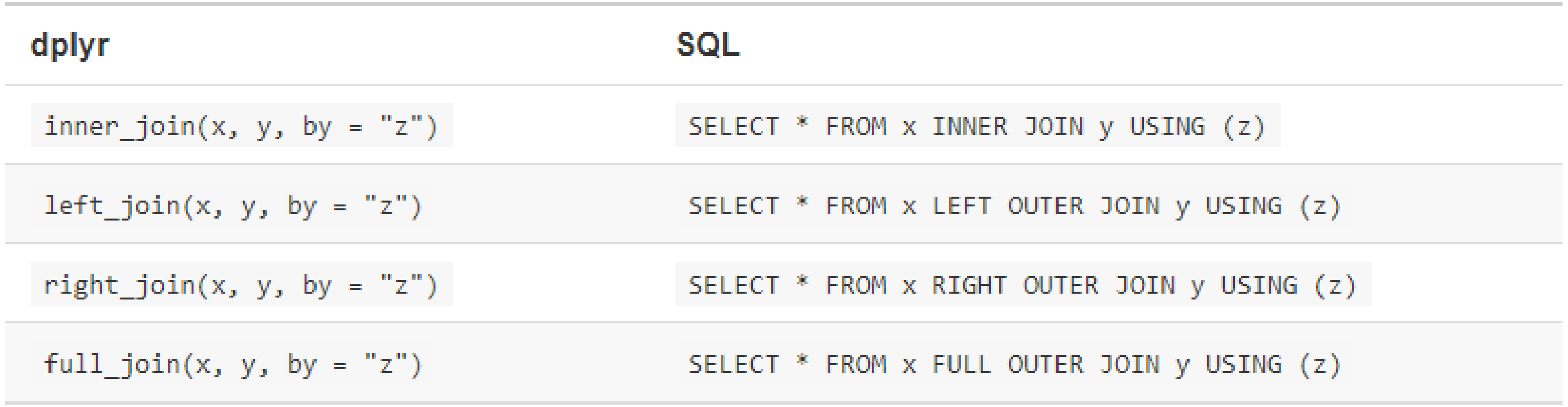
Duplicate keys

```
>  x2 <- data.frame(key1_x = c(1, 2, 2),
                    key2_x = c("a", "a", "b"),
                    val_x = c("y1", "y2", "y3"))

>  y2 <- y2 <- data.frame(key1_y = c(1, 2, 2, 3, 3),
                    key2_y = c("a", "a", "b", "a", "b"),
                    val_y = c("y1", "y2", "y3", "y4", "y5"))

> left_join(x2, y2, c("key1_x" = "key1_y"))

> left_join(x2, y2, c("key1_x" = "key1_y", "key2_x" = "key2_y"))
```

Structured Query Language

Having multiple tables that can be joined together is common in Relational Database Systems (RDBS).

- A common language used by RDBS is Structured Query Language (SQL)



dplyr	SQL
`inner_join(x, y, by = "z")`	`SELECT * FROM x INNER JOIN y USING (z)`
`left_join(x, y, by = "z")`	`SELECT * FROM x LEFT OUTER JOIN y USING (z)`
`right_join(x, y, by = "z")`	`SELECT * FROM x RIGHT OUTER JOIN y USING (z)`
`full_join(x, y, by = "z")`	`SELECT * FROM x FULL OUTER JOIN y USING (z)`

Let’s try it in R…

Scrollytelling with Quarto and Closeread

Scrollytelling

"Scrollytelling" is a web design technique that uses visual and textual elements to tell a story as the reader scrolls through a page

- Closeread examples, and other examples

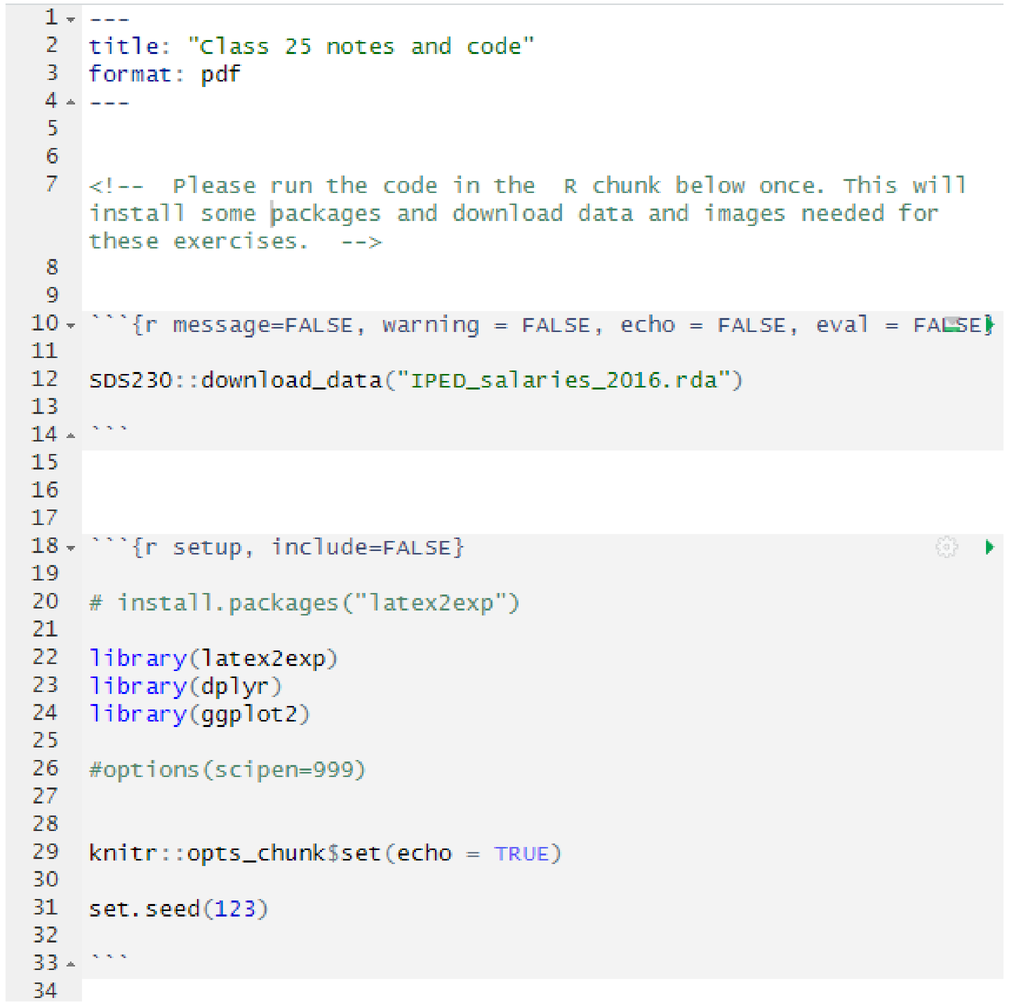
Quarto is an open-source publishing system that combines text, code, and output in one document. It is a successor to R Markdown.

We can creating scrollytelling documents by using the Closeread is a custom format in Quarto

Creating a Closeread Quarto document

To create a Closeread Quarto document, we start by specifying the appropriate quarto header

```
---
title: My Closeread story
format: closeread-html
---
```



~~~
---
title: "Class 25 notes and code"
format: pdf
---


<!--  Please run the code in the  R chunk below once. This will
install some packages and download data and images needed for
these exercises.  -->


```{r message=FALSE, warning = FALSE, echo = FALSE, eval = FALSE}

SDS230::download_data("IPED_salaries_2016.rda")

~~~

```{r setup, include=FALSE}

# install.packages("latex2exp")

library(latex2exp)
library(dplyr)
library(ggplot2)

#options(scipen=999)


knitr::opts_chunk$set(echo = TRUE)

set.seed(123)

```

```
```

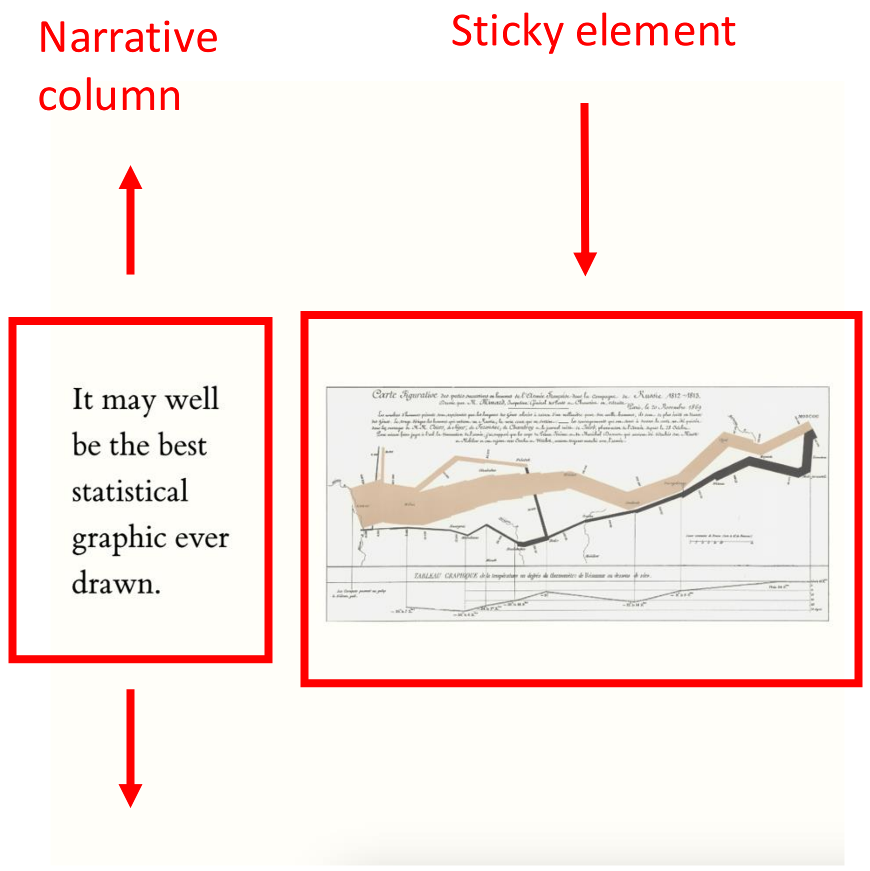
Closeread components

Every Closeread document consists of three components:

1. A **section** of the document flagged as a Closeread section

2. Within the section, a "**sticky**" **element** flagged to appear in the main column of the Closeread section

3. **An element** (often a paragraph of text) that appears in the "narrative column" will serve to trigger the appearance of the sticky element



It may well be the best statistical graphic ever drawn.

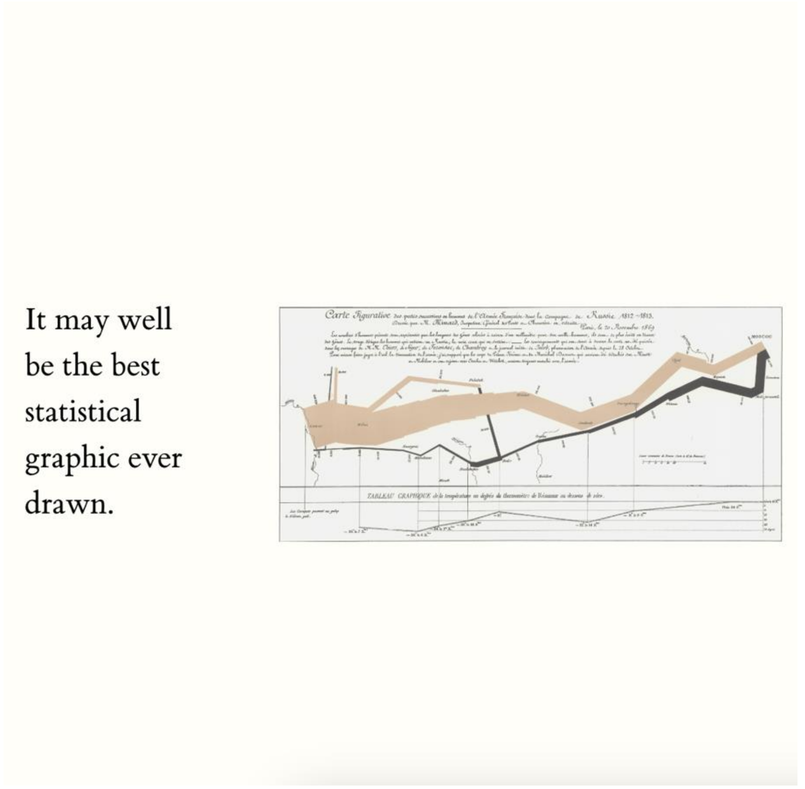
Closeread sections

We can add Closeread sections which specify what is part of our scrolling story (scrolly?)

- Elements outside of a section will appear as a normal Quarto HTML

```
::::{.cr-section}

          Text and images/code go in here…

::::
```

Must have at least 3 colons
Can have more to make sections stand out


It may well be the best statistical graphic ever drawn.

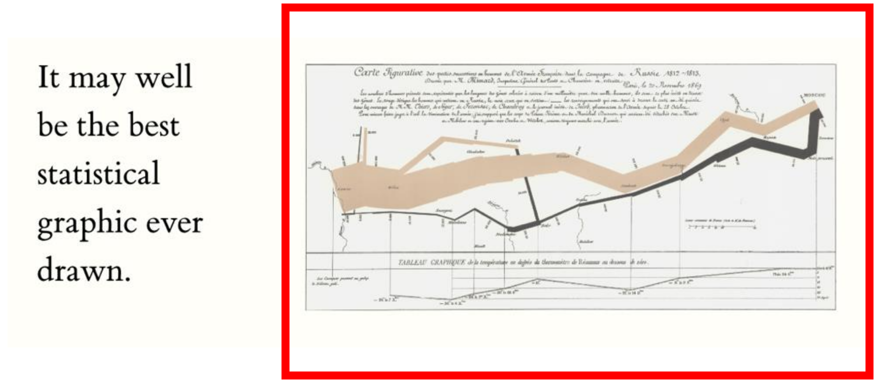
Adding stickies

Elements that are pinned as one scrolls are called ”stickies”

We can add stickies using:

```
:::{#cr-stickyname}
      sticky content
:::
```

Must start with #cr-


It may well be the best statistical graphic ever drawn.

```
:::{#cr-myimage}
   ![](path-to-myimage.png)
:::
```

~~~
:::{#cr-myplot}
   ```{r}
         hist(rnorm(15))
   ```
:::
~~~

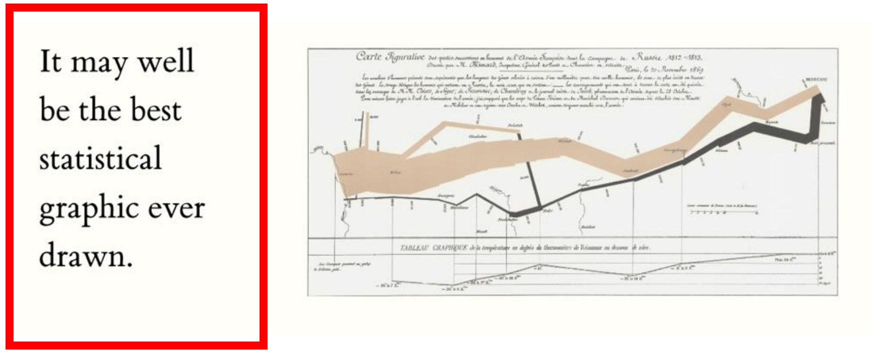
Triggers

Any text (or other elements) in a Closeread section that are not stickies will be placed in the narrative column

You can set a paragraph to trigger focus on a particular sticky by using Quarto’s cross-referencing syntax: @cr-mysticky

It may well be the best statistical grapic every drawn

@cr-myimage


It may well be the best statistical graphic ever drawn.

Let’s now show a histogram @cr-myplot

Focus effects

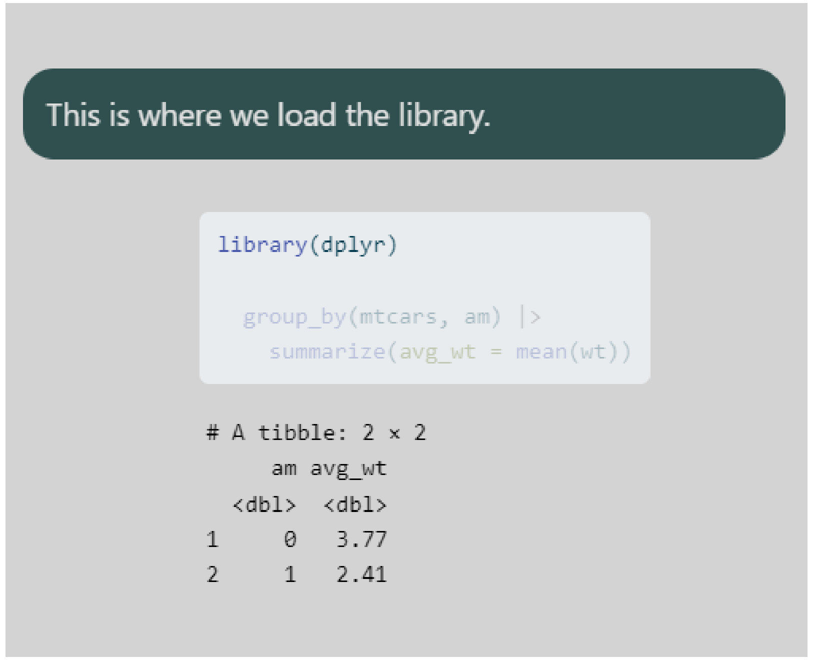
We can add “focus effects” which can do the following:

- Scale, pan, or zoom in on images (or other elements)
- Highlight lines of text

Example:

This is where we load the library. [@cr-dplyr]{highlight="1"}
This calculates summary statistics. [@cr-dplyr]{highlight="2-3"}

~~~
:::{#cr-dplyr}
    ```{r}
    library(dplyr)
    group_by(mtcars, am) |>
          summarize(avg_wt = mean(wt))
   ```
:::
~~~


This is where we load the library.

```
library(dplyr)

  group_by(mtcars, am) |>
    summarize(avg_wt = mean(wt))
```

```
# A tibble: 2 × 2
     am avg_wt
  <dbl>  <dbl>
1     0   3.77
2     1   2.41
```

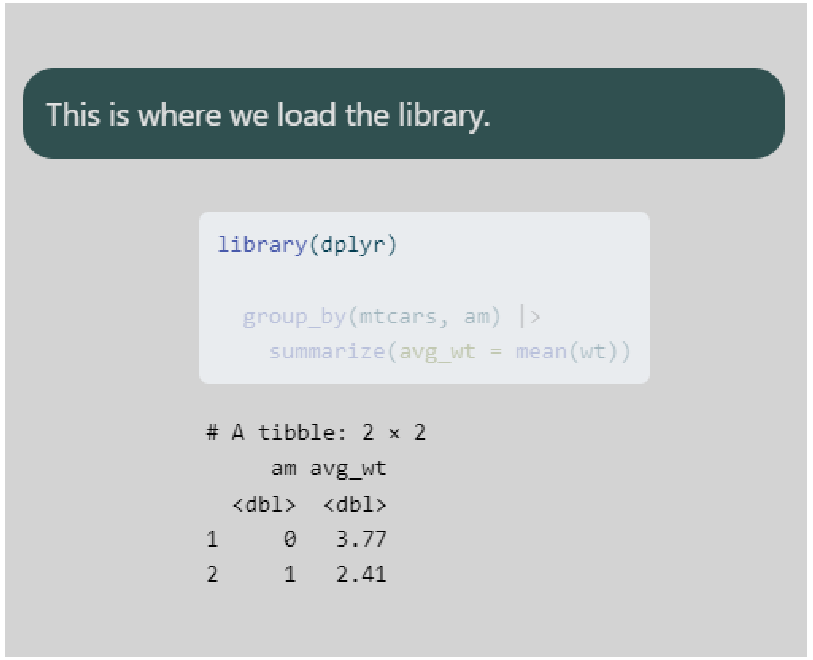
Layouts

We can also change the layout of where the narration text and stickies appear by modifying the Quarto header

```
---
  format:
      closeread-html:
          cr-section:
              layout: "overlay-center"
---
```

Options are:

- sidebar-left sidebar-right
- overlay-right overlay-right overlay-center



This is where we load the library.

```
library(dplyr)

  group_by(mtcars, am) |>
    summarize(avg_wt = mean(wt))
```

```
# A tibble: 2 × 2
     am avg_wt
  <dbl>  <dbl>
1     0   3.77
2     1   2.41
```

Contest

Closeread is running a contest for the best example scrollytelling

Judging criteria will be based on a combination of:

- Educational value or newsworthiness
- Effective use of scrollytelling to deliver the story
- Artistic and design merit
- Technical accomplishment

Awards

- **Honorable mentions**: Posit hex stickers
- **Three runners up**: a free year of Posit Connect Cloud
- **The grand winner**: Posit swag prize pack valued at $200

Deadline: Dec 15

Installing the Closeread Quarto extension

To use Closeread you need to install the Quarto extension by running the following at the terminal tab:

```
quarto add qmd-lab/closeread
```

To get to work you might have to open a new terminal in RStudio using:

- Tools -> Terminal -> New Terminal

Let’s try it in R!