ANOVA continued, text manipulation

Overview

Review and continuation of ANOVAs

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

Text manipulation

- String manipulation with stringr
- If there is time: regular expressions

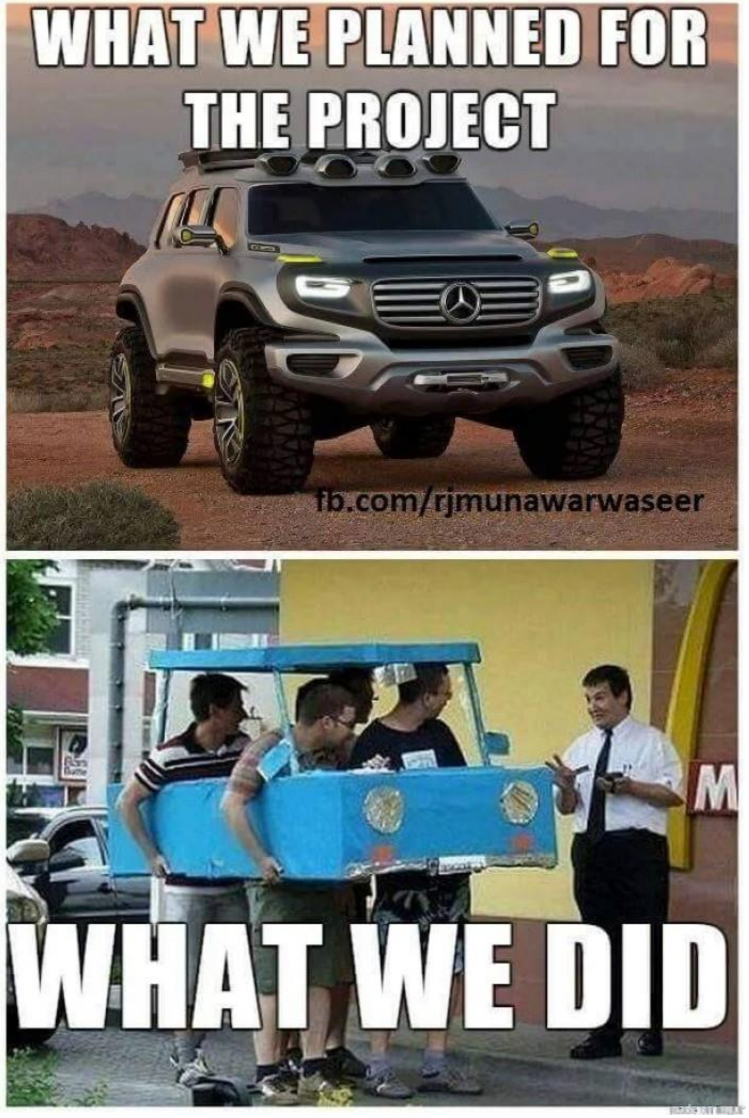
How are final projects going?

Questions from Ed Discussions:

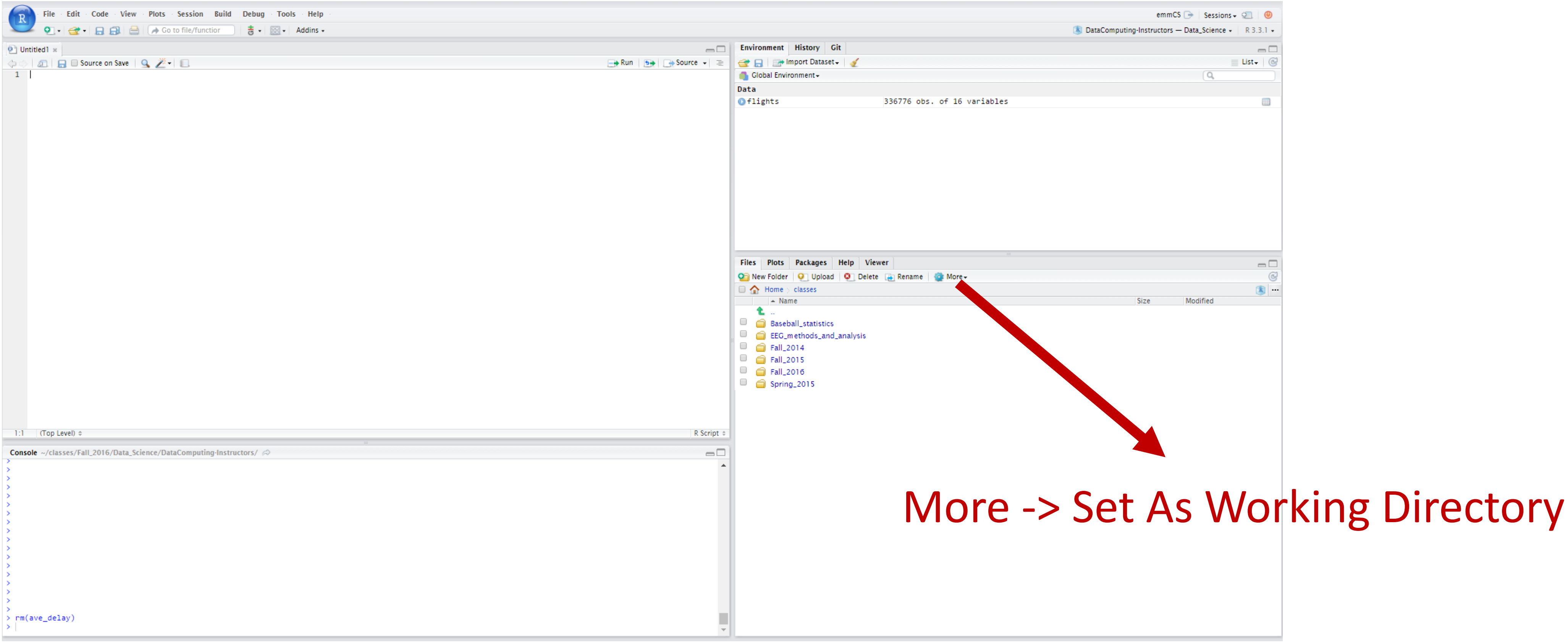
- Can we use built in R functions for tests?
  - Yes!
- If I find a data set I like better than the data set I submitted on P Set 10, could I get it approved and use it for my project?
  - Yes!

Any other questions about the final project?

Note: for your final project, you might want to visualize your data first, and then do modeling/hypothesis testing



Setting your working directory



1. In the files tab, navigate to the directory that contains your data and final project .Rmd file
2. Click More -> Set As Working Directory
3. You should note be able to use read.csv() to load data that is in you working directory

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:

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

ε_{ijk}: Random error for the ijk^{th} data point

y_{ijk}: i^{th} response when:
- factor A has level j
- Factor B has level k

μ: Overall mean

α_j: Main effect for factor A at level j

β_k: Main effect for factor B at level k

γ_{jk}: Specific interaction for j^{th} level of A and k^{th} level of B

Two-way ANOVA hypotheses

Main effect for A

H_0: $\alpha_1 = \alpha_2 = \ldots = \alpha_J = 0$

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

Main effect for B

H_0: $\beta_1 = \beta_2 = \ldots = \beta_K = 0$

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

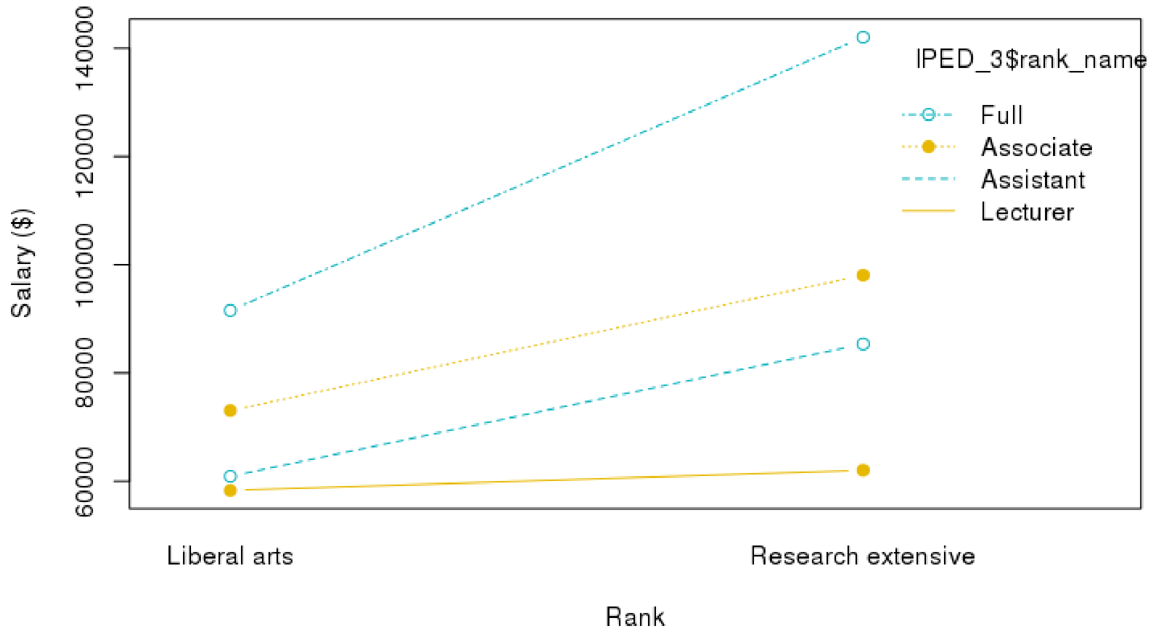
γ_{jk} : is the interaction between level j of factor A, and level k of factor B.

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

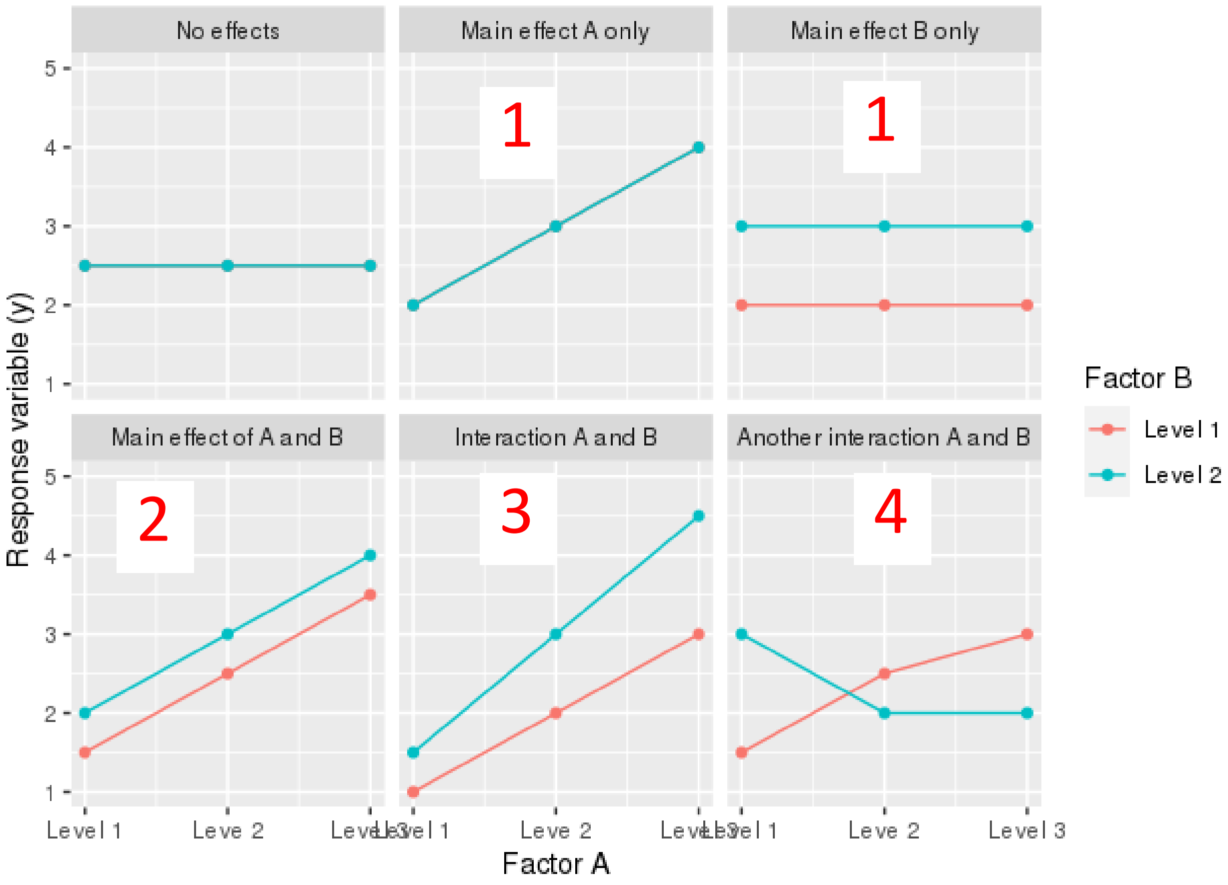
Interaction plots

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

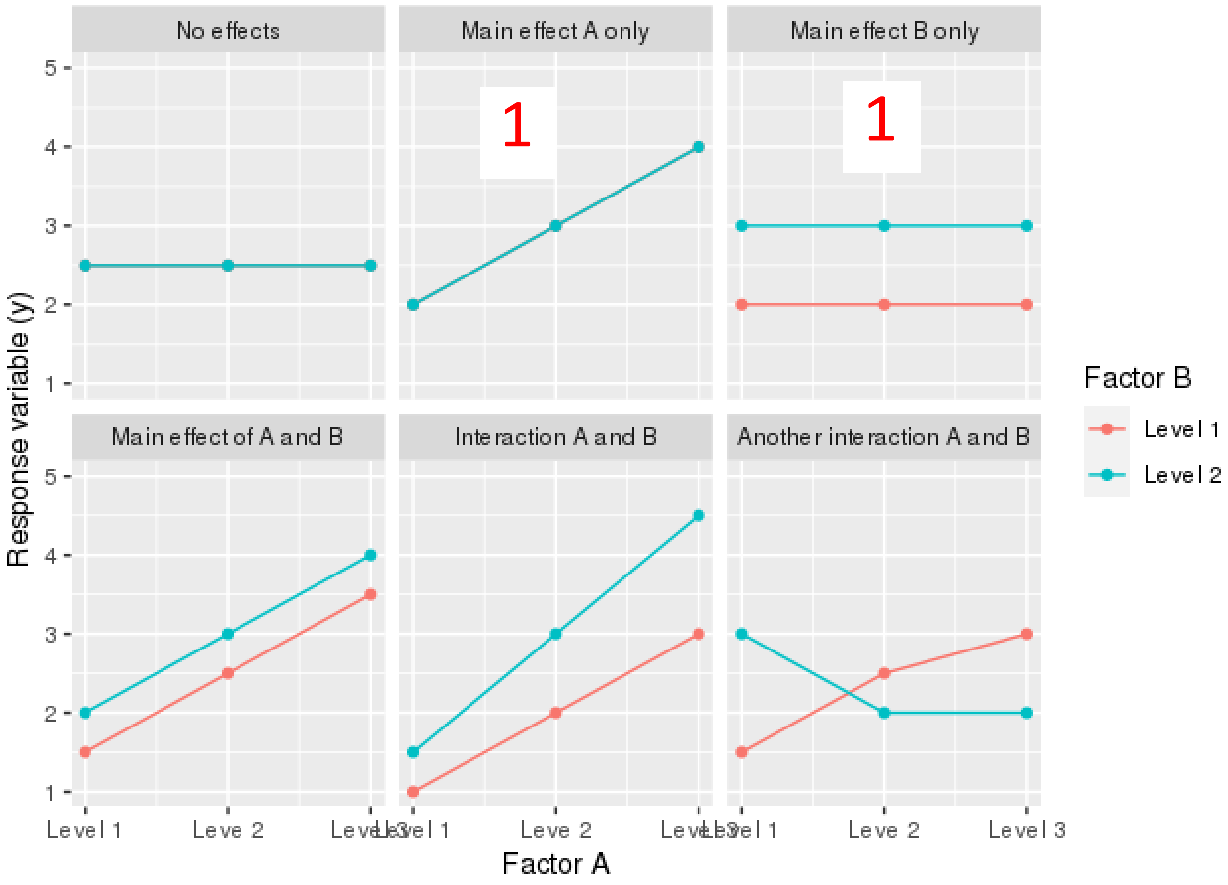


Interpreting interaction through interaction plots



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

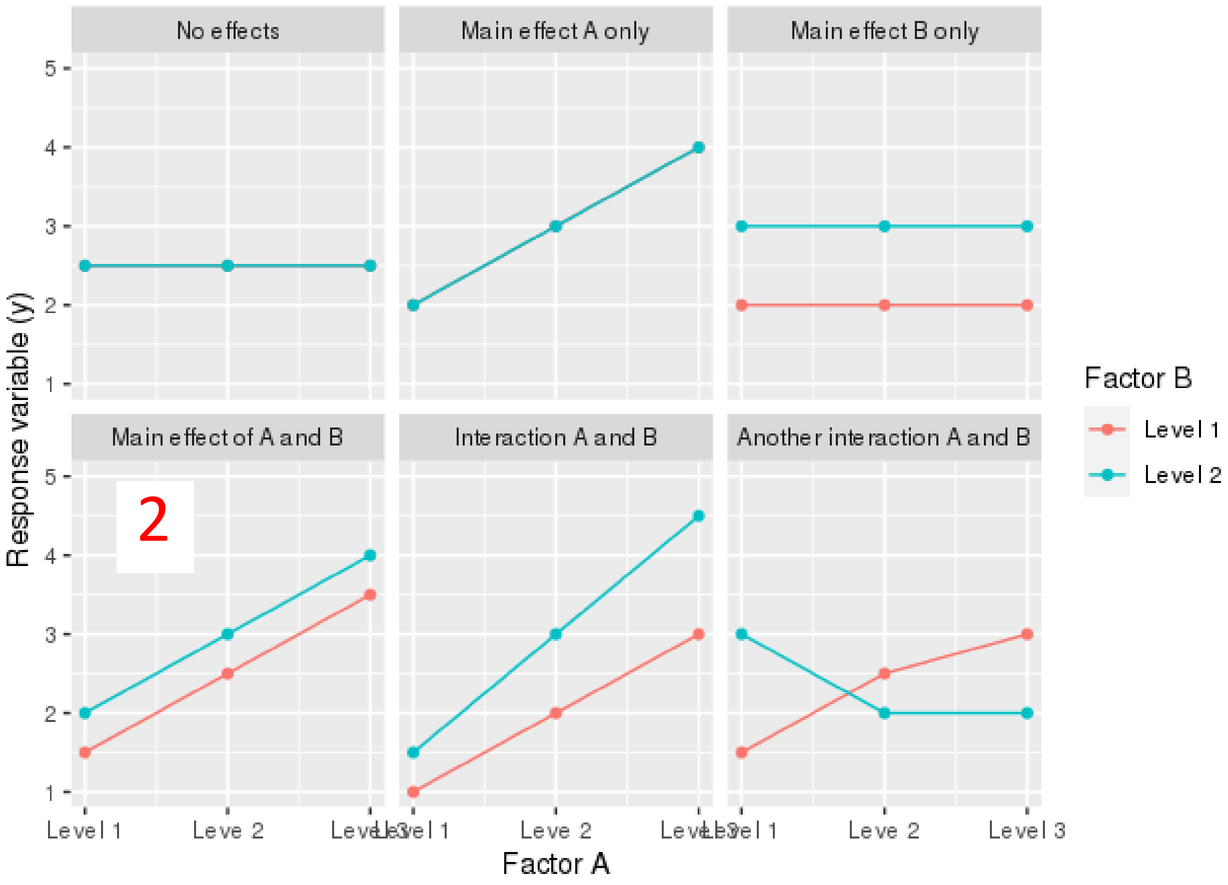
Interpreting interaction through interaction plots



What are examples we have seen in class of 1 (main effect only in one factor)?

$$y_{ijk} = \mu \quad + \quad \beta_k \quad + \quad \varepsilon_{ijk}$$

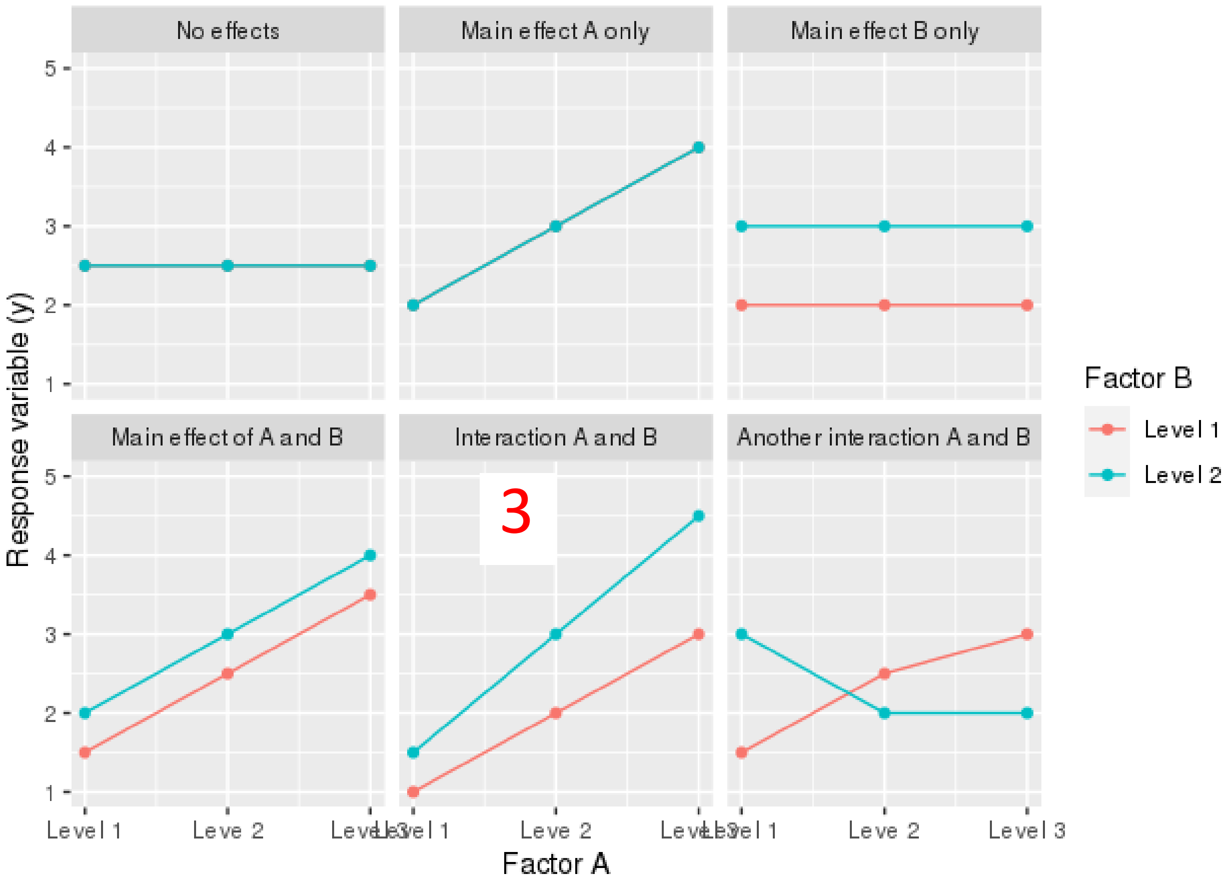
Interpreting interaction through interaction plots



What are examples we have seen in class of 2 (main effect in both factors, no interaction)?

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

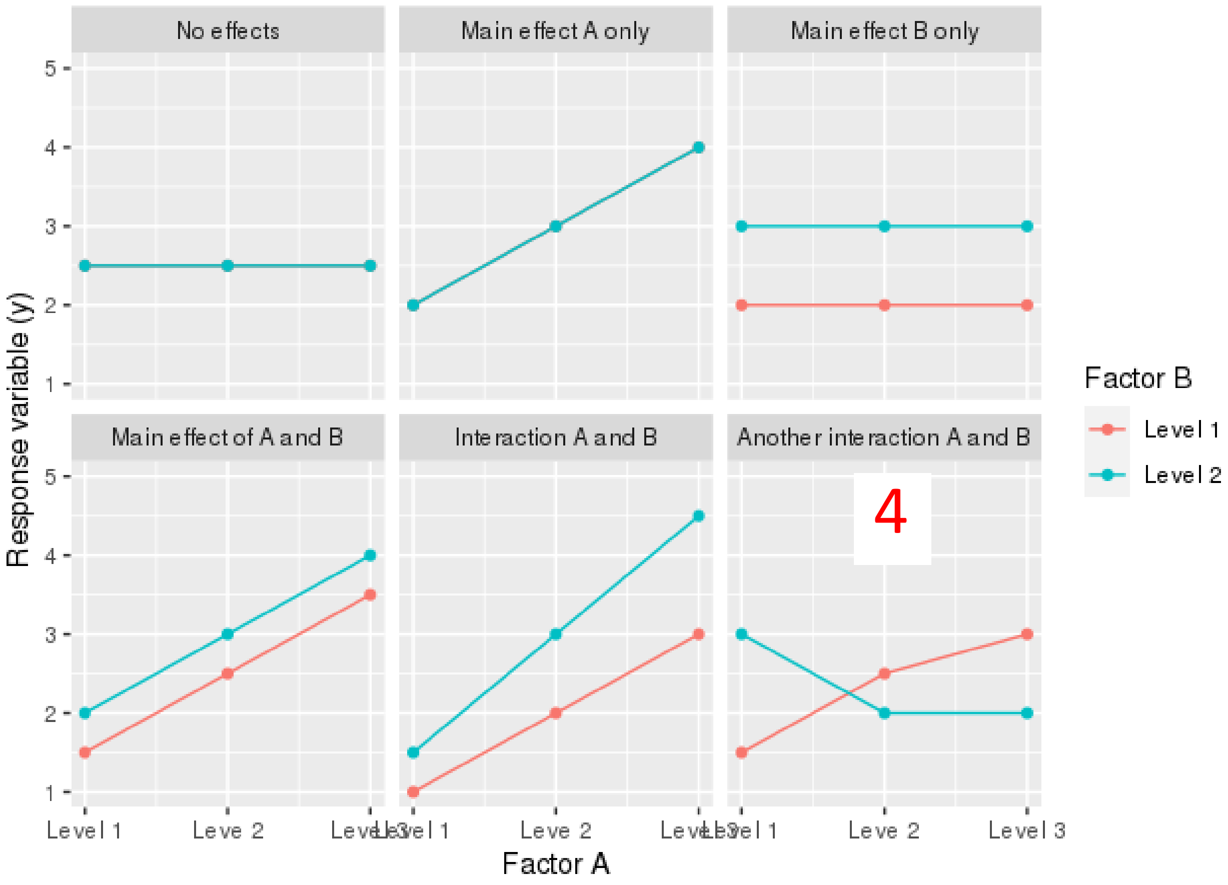
Interpreting interaction through interaction plots



What are examples we have seen in class of 3 (main effects and interaction)?

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

Interpreting interaction through interaction plots



What are examples we have seen in class of 4 (reverse ordering of effects with interaction)?

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

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

For unbalanced designs, there are different ways 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.

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

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

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

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

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 or caramel sauce.

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

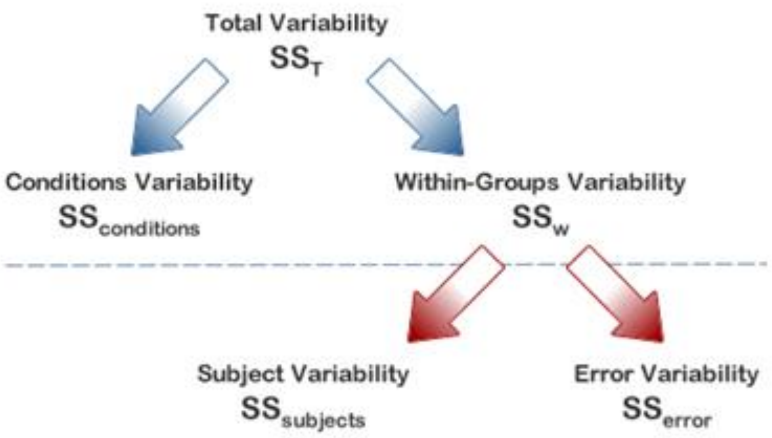
Within subjects experiment: each person in the experiment gives ratings for all three toppings.

- Run a repeated measures ANOVA

Repeated measures ANOVA

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

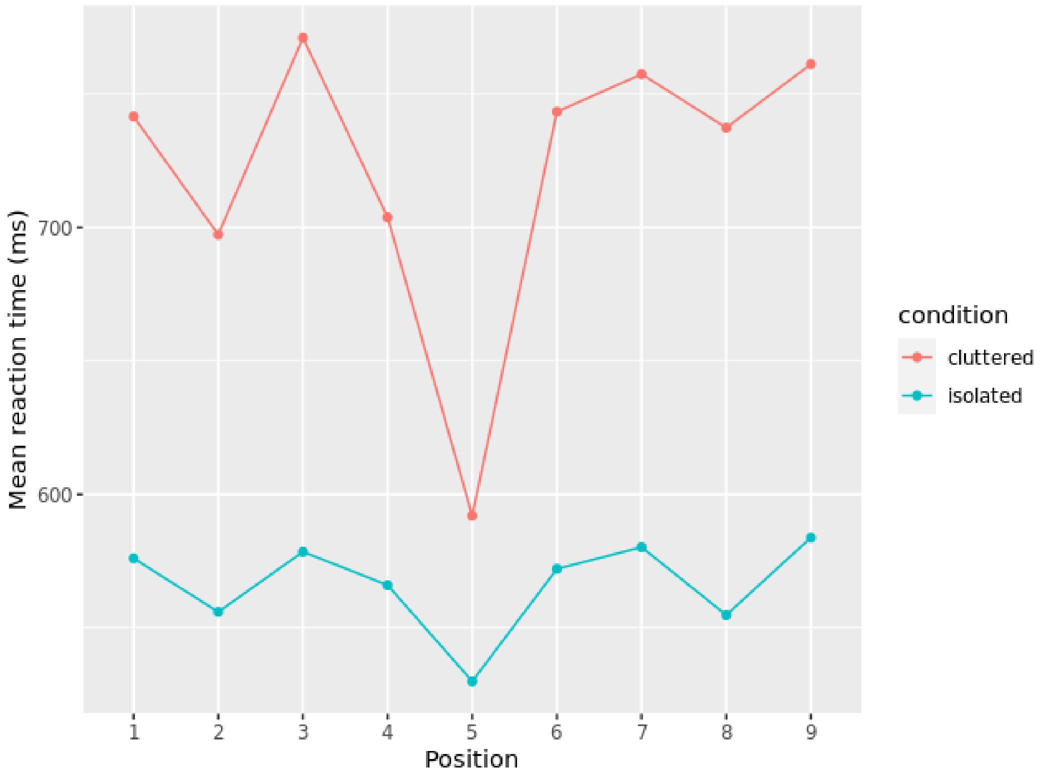
- This is a generalization of a paired t-test to more than two population means

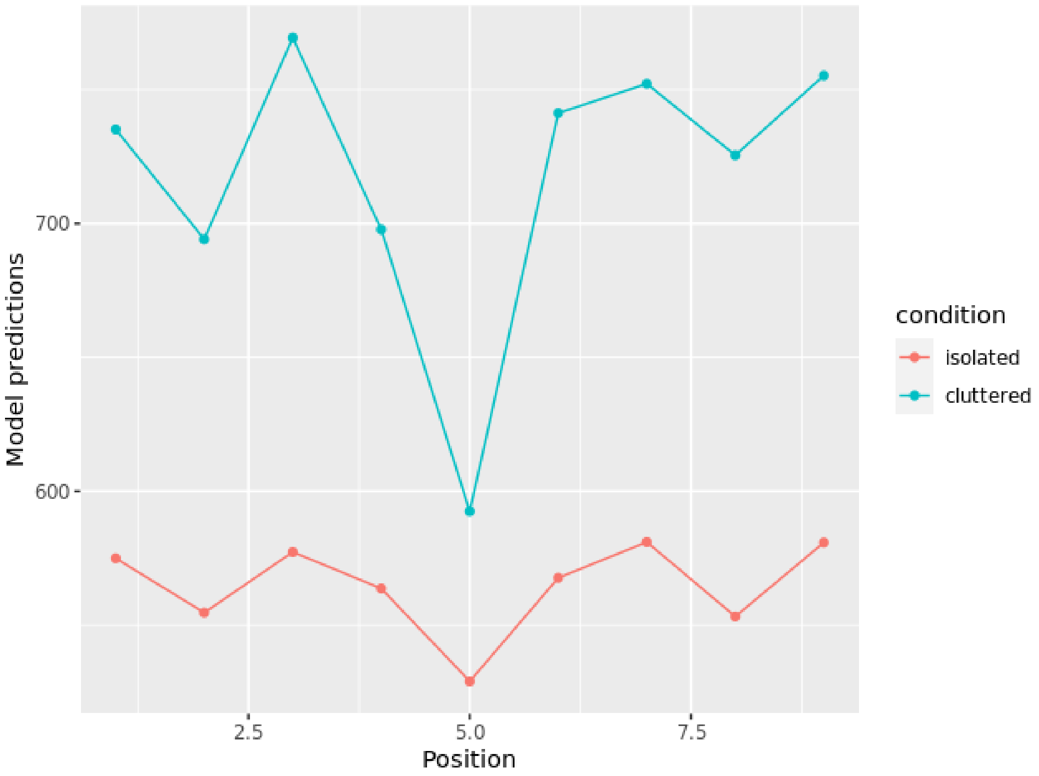


To run a repeated meseasures ANOVA, we use a factor called ID that has a unique value for each observational unit

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

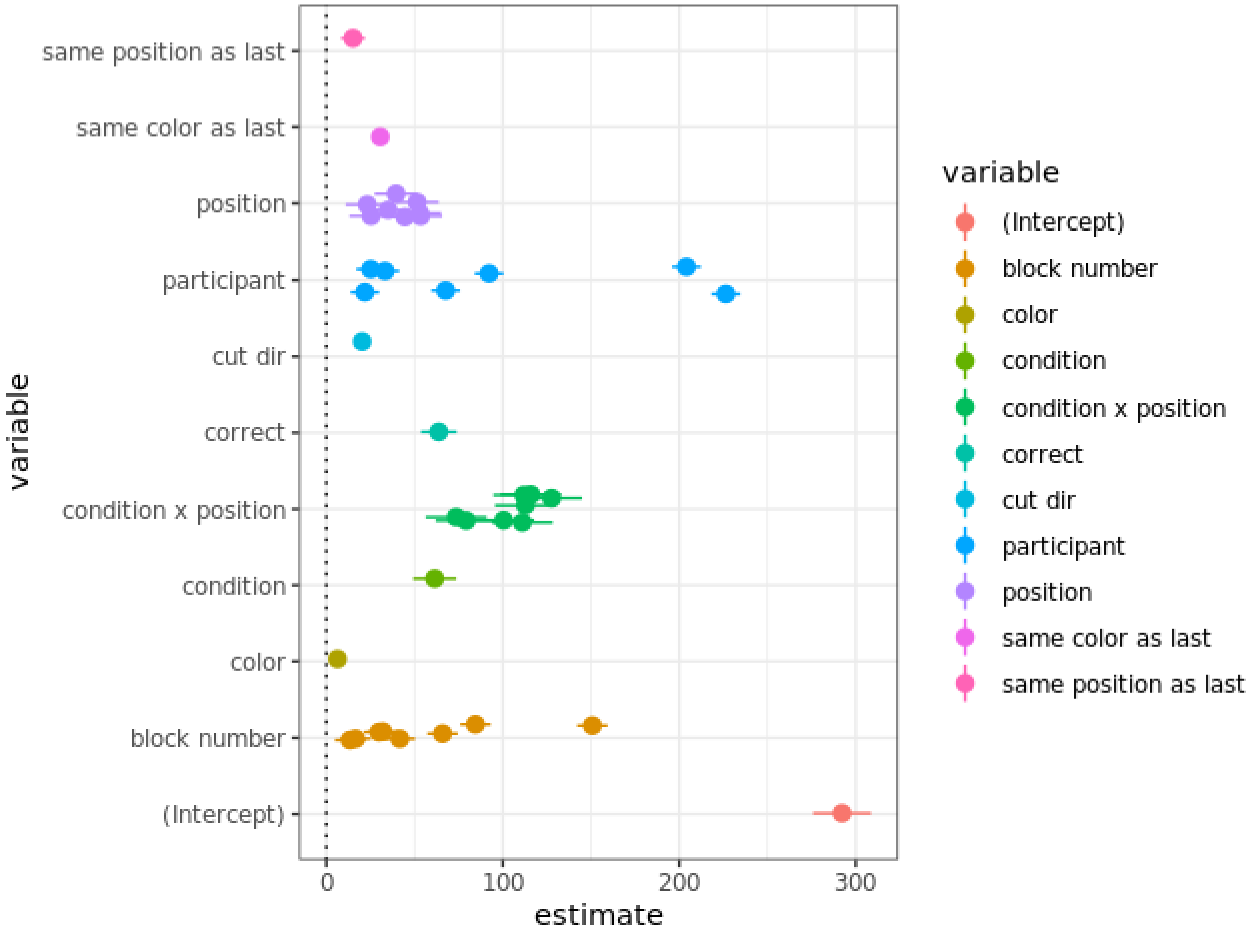
The ANOVA model – popout data

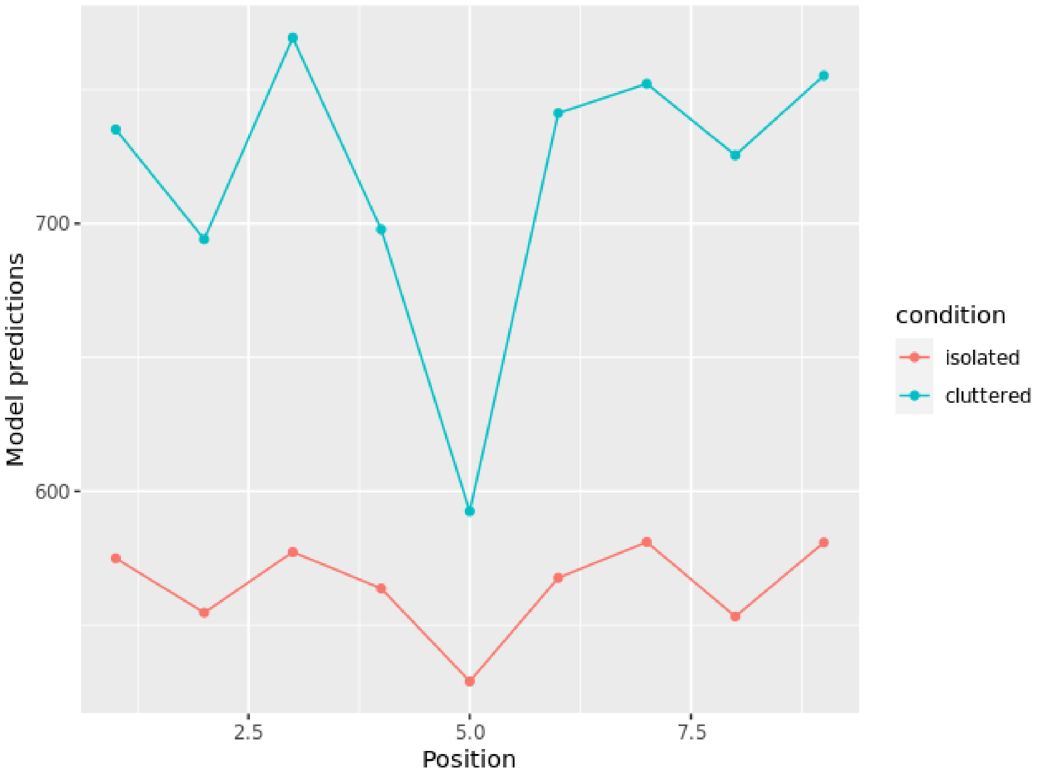




aov(reaction_time ~ condition + position + color + cut_dir + correct + block_number + participant + same_position_as_last + same_color_as_last + position * condition, data = popout_data)

The ANOVA model – popout data

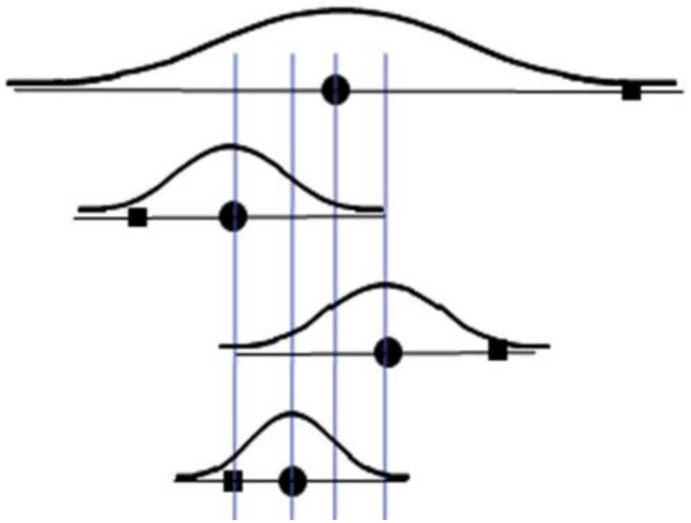




aov(reaction_time ~ condition + position + color + cut_dir + correct + block_number + participant + same_position_as_last + same_color_as_last + position * condition, data = popout_data)

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.

- We then just estimate a mean and standard deviation for the underlying population, rather a separately ID for each participant.
  - 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 these topics in R…

Text manipulation

80% of a Data Scientists time is cleaning data

- Text manipulation is a big part of cleaning data

20% of a Data Scientists time is complaining about cleaning data

Text manipulation in R

Base R has a number of text manipulation functions

```
> tolower("Hey")
```

stringr is the tidyverse version

- Does many of the same things with a more consistent syntax (e.g., all functions start with str_)

```
> str_to_lower("STOP YELLING")
```

We will focus on stringr because it’s a bit of an improvement over base R's functions

```
> library(stringr)
```

Let's try the rest in R...

str_trim and str_pad

str_trim removes leading and trailing whitespace.

- Similar to base R: strtrim()

Example:

```
> str_trim("    What a mess      ")
```

str_pad adds extra whitespace whitespace

Example:

```
> str_pad("Let's make it messier", 50, "right")
> str_pad(1:11, 3, pad = 0)   # useful for adding leading 0's
```

str_sub

Returns a substring from the original string

- str_sub("String", start.m, end.n)
- Equivalent to base R: substr()

Examples:

```
> str_sub("What a mess", 6, 11)

> fruits <- c("apple", "pineapple", "Pear", "orange",  "peach", "banana")
> str_sub(fruits, 2, 4)
```

str_c

Combines a number of strings together

- Equivalent to base R: paste()

Examples:

Make sure there is a space here

```
> str_c("What", "a", "mess", sep = " ")
```

```
# we can also concatenate values in a vector
> vec_words <- c("What", "a", "mess")
> str_c(vec_words, collapse = " ")
```

Let's download a web page

```
>  base_name <- "https://www.foxnews.com/politics/biden-deliver-unscheduled-speech-capitol"

>  article_name <- "politics.html"

> download.file(base_name, article_name)

> viewer <- getOption("viewer")

> viewer(article_name)
```

str_length

Returns the number of characters in the string

- Equivalent to base R: nchar()

Examples:

```
> str_length("What a mess")

> article_size <- file.info(article_name)$size    # size of the article in bytes

# read the whole article as a single string
> the_article <- readChar(article_name, article_size)

> str_length(the_article)    # size of the article as a string
```

str_replace_all

Takes a string, and replaces every instance of substring with a new string

- > str_replace("String", "old", "new")
- Equivalent to base R: gsub()

Example:

```
> article2  <- str_replace_all(the_article,  "Biden", "Sleepy Joe")

> write(article2, "sleepy_article.html")

> viewer("sleepy_article.html")
```

str_split

Splits a single string into a list of strings

- > str_split("String", "split pattern")
- Equivalent to base R: strsplit()

Make sure there is
a space here

Examples:

```
> list_of_strings <- str_split("What a mess",  " ")
> vector_of_strings <- unlist(list_of_strings)
> vector_of_strings[3]

> article_vec <- unlist(str_split(the_article, " "))
```

str_extract

Extract a pattern from a string

- > str_extract("String", "pattern")
- Equivalent to base R: sub()

Examples:

```
> str_extract(fruits,  "apple")
```

str_detect

Check to see if a pattern occurs in a string

- > str_detect("String", "pattern")
- Equivalent to base R: grepl()

Examples:

```
> str_detect(fruits,  "apple")
```

```
# can you tell how many times Biden was mentioned in the article?
> sum(str_detect(article_vec, ”Biden"))
```

Regular expressions

Regular expressions are string that allow you find more complex patterns in strings

For example:

- The character "^" indicates the beginning of a string
- The character "$" indicates the end of a string
- The expression "[Pp]" indicates "P" or "p"

```
# what do these expressions do?
> str_detect(fruits, "e$")
> str_detect(fruits, "^[Pp]")
```

The following are special regular expression characters that are reserved:

. * \ $ { } [ ] ^ ?

Regular expressions

. (period) matches any single character

- > str_detect(c("mess", "mass", "miss"), "m.ss")

* means match 0 or more of the preceding character

- > str_detect(c("xz", "xyz", "xyyz", "xyyyz"), "xy*z")

+ means match 1 or more of the preceding character

- > str_detect(c("xz", "xyz", "xyyz", "xyyyz"), "xy+z")

what will the following match?

- > str_detect(fruits, "^a.*e$")

what about if the ^ was removed?

Regular expressions

[ ] means match anything in the range inside the braces

- > str_detect(fruits, "^[a-o]")
- > str_detect(c("chimp", "champion", "chomp"), "ch[aio]mp")

Note: if the ^ appears inside square braces it means not

- > str_detect(fruits, "^[^a-o]")

() groups things together, useful in combination with {}

{num} means repeat the preceding sequence num times

- > str_detect(fruits, "(an){2}")
- > str_extract(fruits, "(an){1,}")

Example

```
strings <- c(
        "apple",
        "219 733 8965",
        "329-293-8753",
        "Work: 579-499-7527",
        "Home: 543.355.3679"
)

phone <- "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"

str_extract(strings, phone)
```

Escape sequences

In regular expressions a period (.) means any character

- So how can you detect if a period is in a string?

Escape sequences in R start with two slashes \\ and cause the next character to be treated literally rather than as a special character

- To match a period we use \\.   [.] also works
- To match a $ symbol we use \\$

Extract the amounts of money and dollar sign from this string (use str_extract_all)

- `> the_string <- c("Joanna has $100 and Chris has $0")`
- `> str_extract_all(the_string, "\\$[0-9]{1,}")`

Character classes

Other special characters are also designated by using a double slash first

- \\s space
- \\n new line or also \\r
- \\t tab

```
# get 6 characters prior to the end of a line in the_article
> str_extract_all(the_article, ".{6}\\n")
```

```
# all ending html tags
> end_tag <- str_extract_all(the_article, "</[A-z]{1,}>\\n")
> lapply(end_tag, str_replace, "\n", "")
```


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP