Data Visualisation with `ggplot2`

Communicating your data

Daniela Palleschi

2024-06-03

Table of contents

Learning objectives

- visualise variable distributions
- visualise summary statistics
- save figures as `rds` or as figures

Load packages and data

```
# load packages
pacman::p_load(tidyverse, here)
```

```
# load data
df_lifetime <- readr::read_csv(here::here("data/tidy_data_lifetime_pilot.csv"),
                               # for special characters
                               locale = readr::locale(encoding = "latin1")
                               ) |>
  mutate_if(is.character,as.factor) |> # all character variables as factor
  filter(type == "critical", # only critical trials
         px != "px3") # this participant had lots of 0's for some reason
```

Summary of first-fixation times

- this code is from the previous topic

```
# compute summary
summary_ff <- df_lifetime |>
  filter(region=="verb") |>
  group_by(condition, lifetime, tense) %>%
  summarise(N = n(),
            mean.ff = mean(ff, na.rm = T),
            sd = sd(ff, na.rm = T)) %>%
  # compute standard error, confidence intervals, and lower/upper ci bounds
  mutate(se = sd / sqrt(N),
         ci = qt(1 - (0.05 / 2), N - 1) * se,
         lower.ci = mean.ff - qt(1 - (0.05 / 2), N - 1) * se,
         upper.ci = mean.ff + qt(1 - (0.05 / 2), N - 1) * se) |>
  ungroup()
```

Plotting reading times

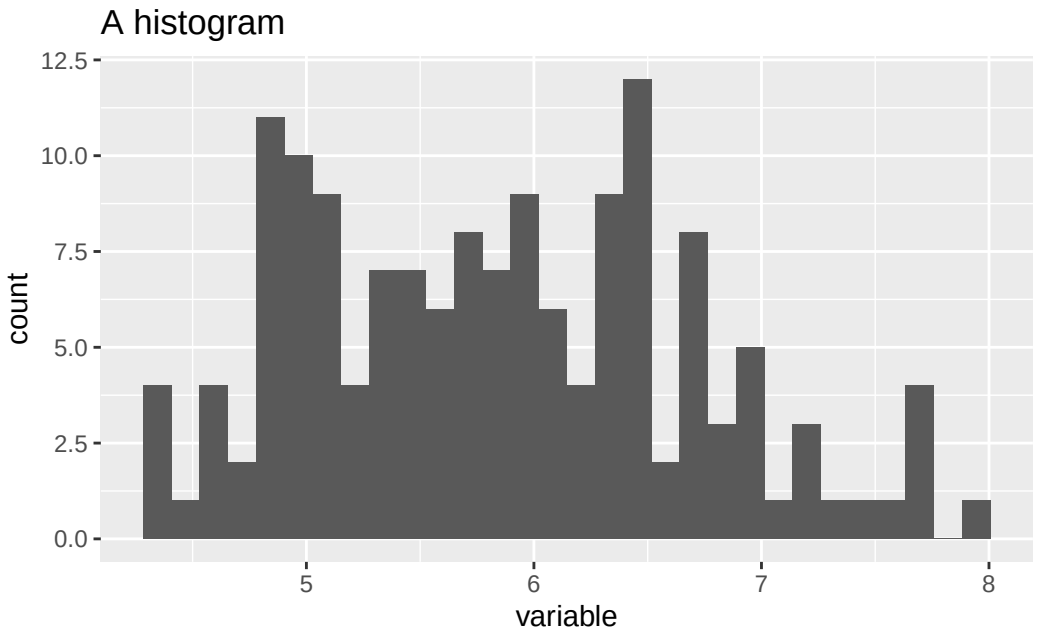
- reading times are (usually) continuous variables
  - as are e.g., reaction times
- they are truncated at 0, meaning they cannot have negative values
  - because of this, they tend to have a *skewed distribution*

Plots with `ggplot2`

- `ggplot2` is part of the tidyverse (like `dplyr`)
  - uses a *layered grammar of graphics*
  - i.e., we build layers

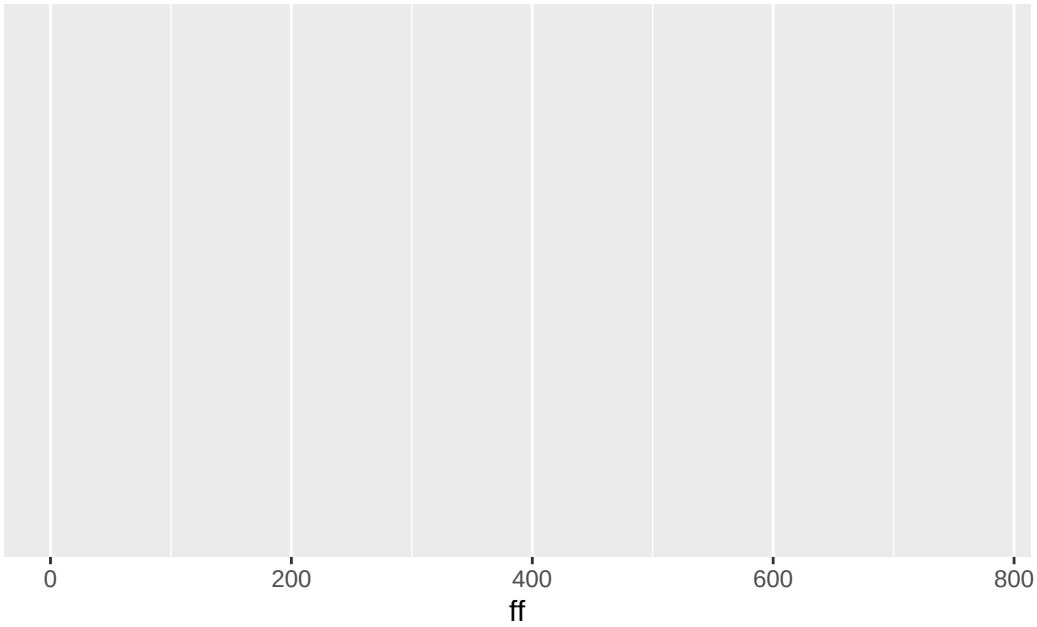
An example: histogram

```r
iris |>
  ggplot(aes(x=Sepal.Length)) +
  geom_histogram() +
  labs(title = "A histogram",
       x = "variable")
```



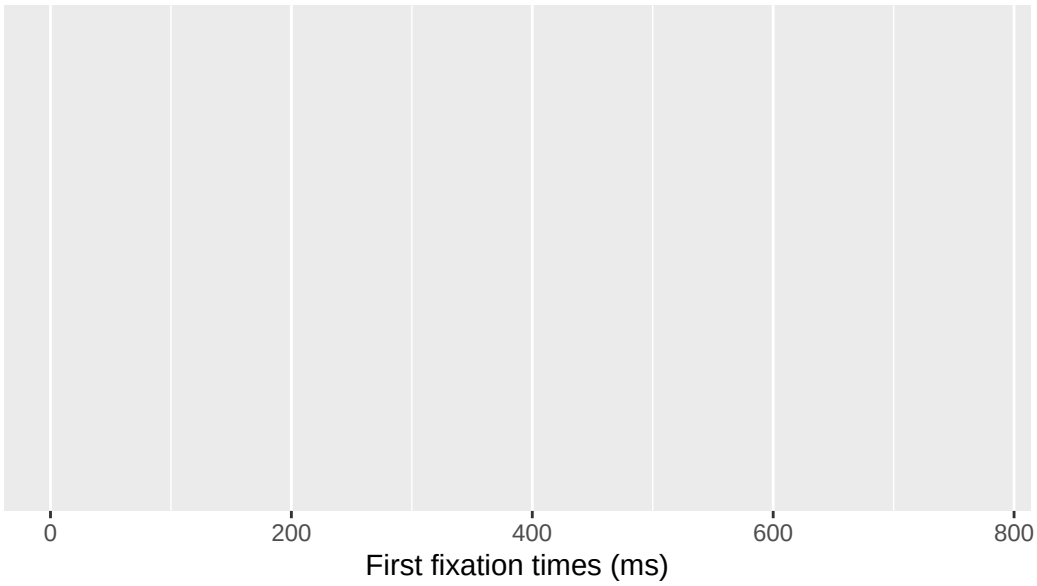
Start layering

```r
df_lifetime |> ggplot(aes(ff)) # aes = 'aesthetic'
```



Add labels

```
df_lifetime |> ggplot(aes(ff)) +
  labs(title = "Histogram of first fixation times",
       x = "First fixation times (ms)")
```



Add

```
df_lifetime |> ggplot(aes(ff)) +
  labs(title = "Histogram of first fixataion times",
       x = "First fixation times (ms)") +
  geom_histogram()
```

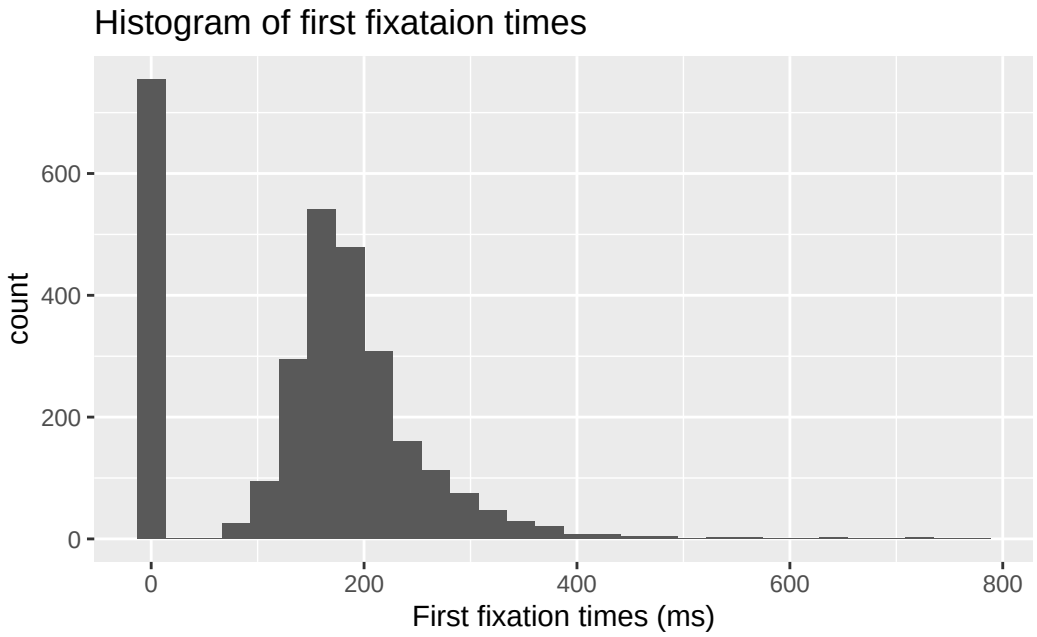


Figure 1: Distribution of first fixation times at the verb region (raw milliseconds)

Add condition

```
df_lifetime |> ggplot(aes(ff, fill = condition)) +
  labs(title = "First fixataion times at the verb region",
       x = "First fixation times (ms)") +
  geom_histogram()
```

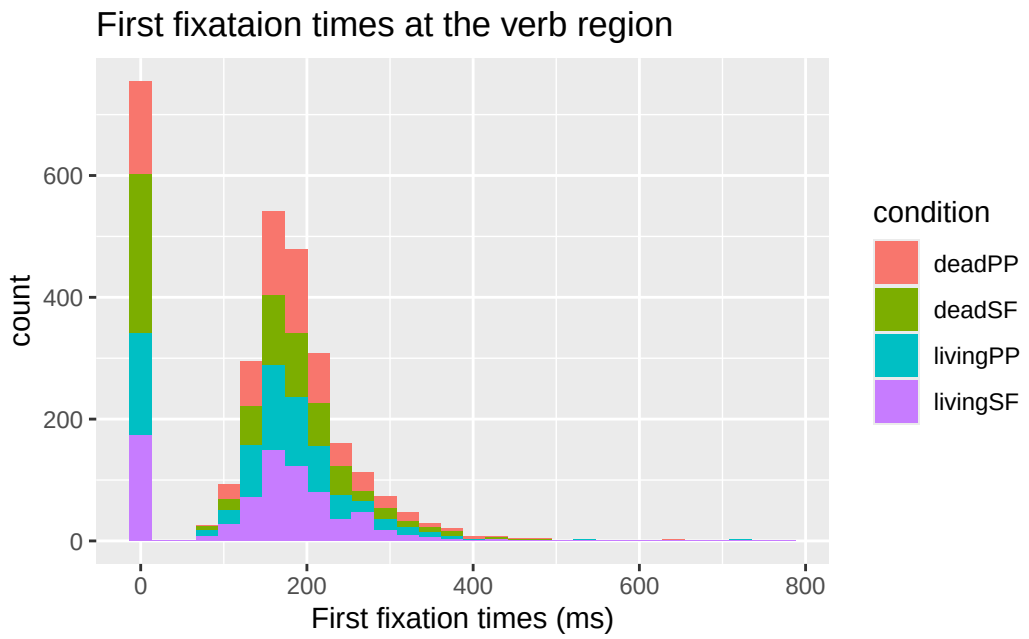


Figure 2: Distribution of first fixation times at the verb region (raw milliseconds)

The colour here is STACKED!! i.e., not layered. Notice the distribution doesn't change from all grey to coloured

Customisation

- we can add arguments to our geoms
  - e.g., transparency: `alpha =` takes a value between 0 to 1
- we can use `theme()` to customise font sizes, legend placement, etc.
- tehre are also popular preset themes, such as `theme_bw()` and `theme_minimal()`

`theme_bw()`

```
df_lifetime |> ggplot(aes(ff, fill = condition)) +
  labs(title = "Histogram of first fixataion times",
       x = "First fixation times (ms)") +
  geom_histogram(alpha=.5) +
  theme_bw()
```

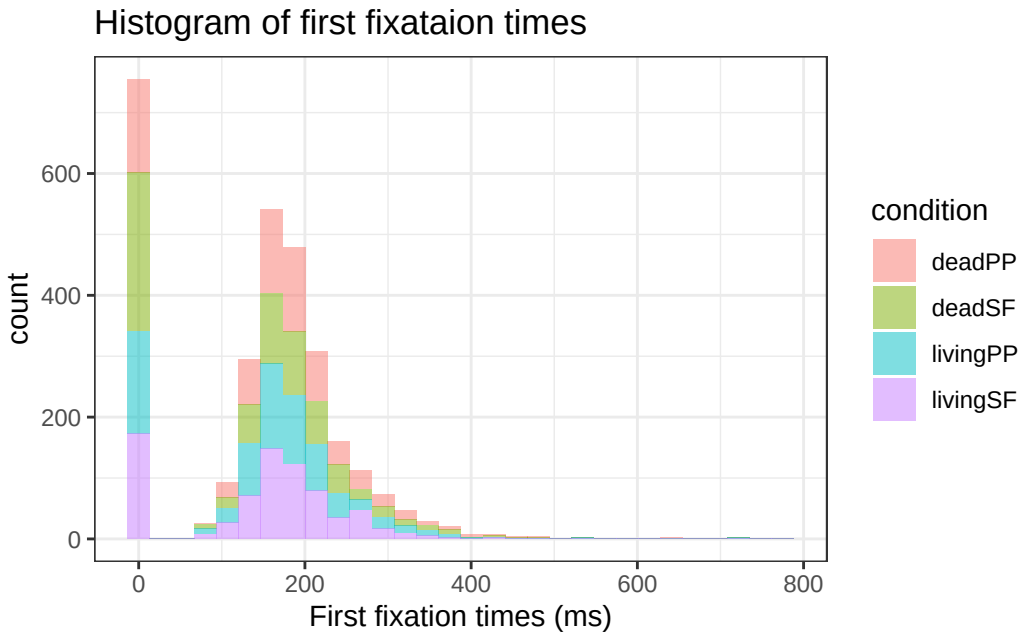


Figure 3: Distribution of first fixation times at the verb region (raw milliseconds).

theme_minimal()

```
df_lifetime |> ggplot(aes(ff, fill = condition)) +
  labs(title = "Histogram of first fixataion times",
       x = "First fixation times (ms)") +
  geom_histogram(alpha=.5) +
  theme_minimal()
```

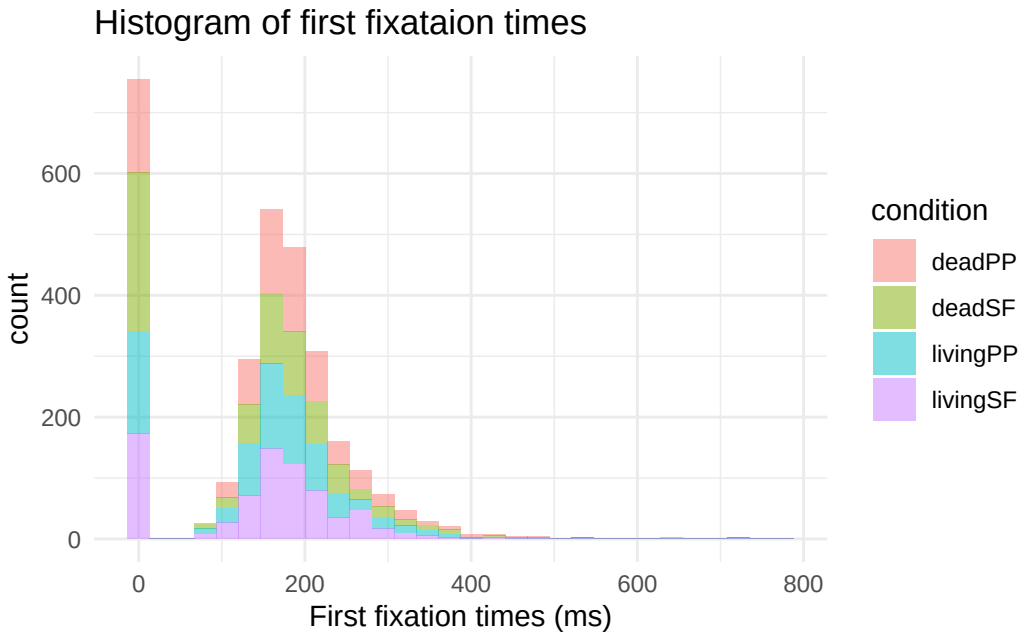
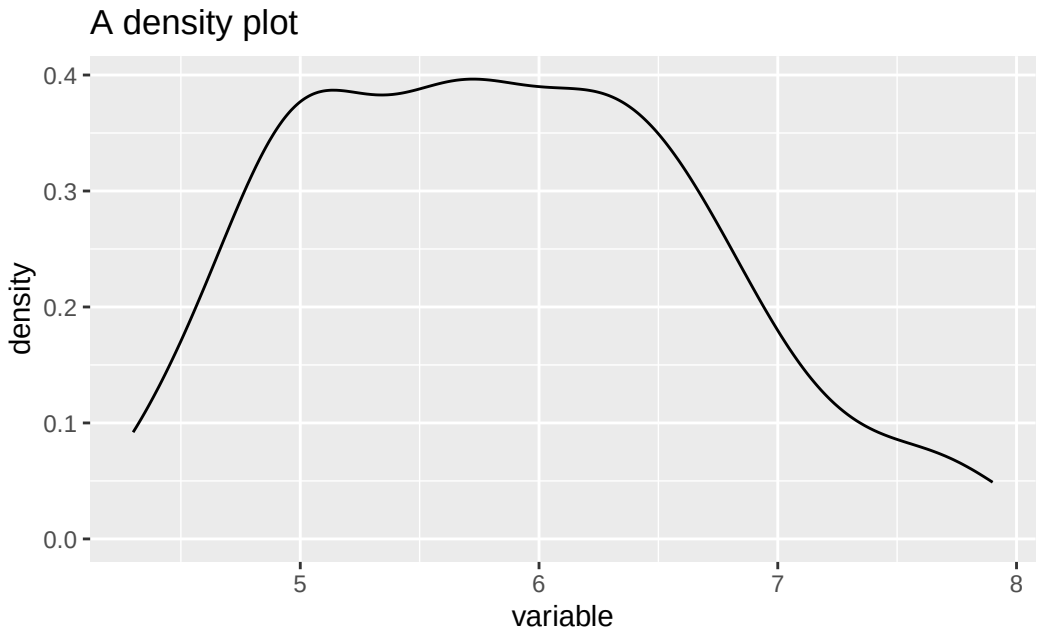


Figure 4: Distribution of first fixation times at the verb region (raw milliseconds).

Distributions

- show the distribution of observations
  - so we can see where the data are clustered
  - and eyeball the shape of the distribution
- we already saw the histogram, which shows the number of observations per variable value
- **density plots** are another useful plot for visualising distributions



Density plots

- below I just replaced `geom_histogram()` with `geom_density()`
  - I also filtered the data to include only values of `ff` above 0
- what is plotted along the y-axis? how does this differ from a histogram?

```
df_lifetime |>
  filter(ff > 0) |>
  ggplot(aes(ff)) +
  labs(title = "Histogram of first fixataion times",
       x = "First fixation times (ms)") +
  geom_density() +
  theme_minimal()
```

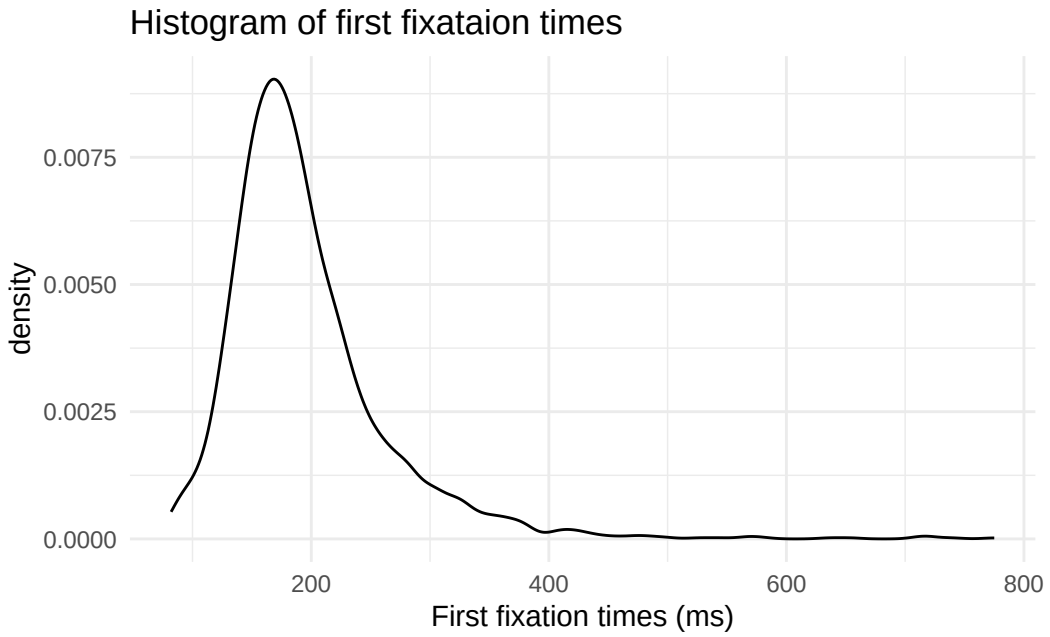


Figure 5: Distribution of first fixation times at the verb region (raw milliseconds).

Grouped density plots

- just like with histograms, we can look at the density plots of different subsets of the data with `aes(fill = )`
  - like region

```
df_lifetime |>
  filter(ff > 0) |>
  ggplot(aes(ff, fill = region)) +
  labs(title = "Histogram of first fixataion times",
       x = "First fixation times (ms)") +
  geom_density(alpha=.5) +
  theme_minimal()
```

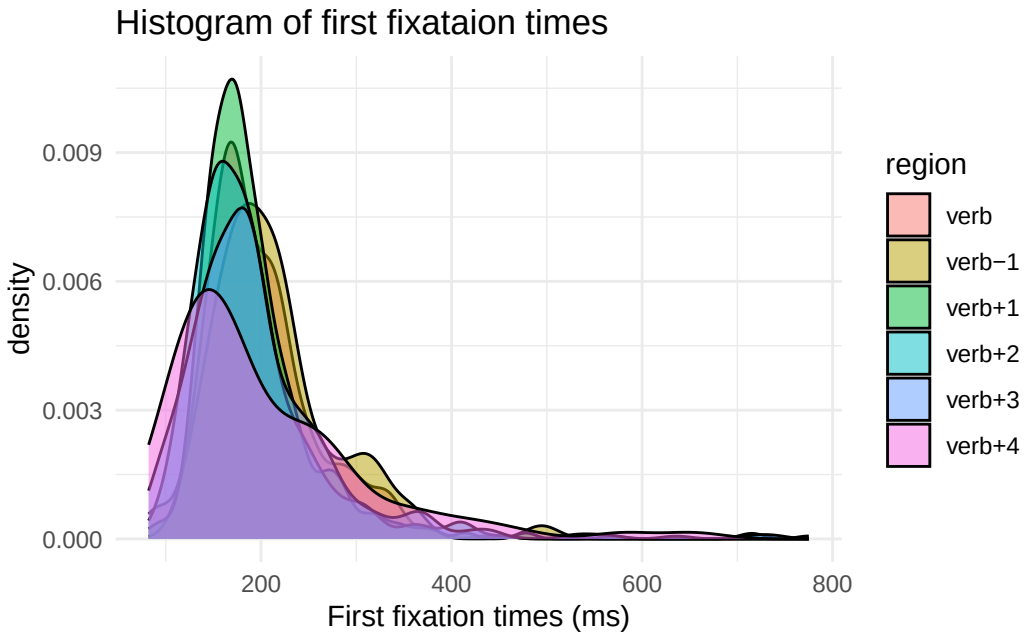


Figure 6: Distribution of first fixation times at the verb region (raw milliseconds).

`facet_grid()`

- there are a lot of overlapping density curves, let's try to separate them with `facet_grid(x~y)`

```
df_lifetime |>
  filter(ff > 0) |>
  ggplot(aes(ff, fill = region)) +
  facet_grid(.~region) +
  labs(title = "Density plot of first fixataion times by region",
       x = "First fixation times (ms)") +
  geom_density(alpha=.5) +
  theme_bw()
```

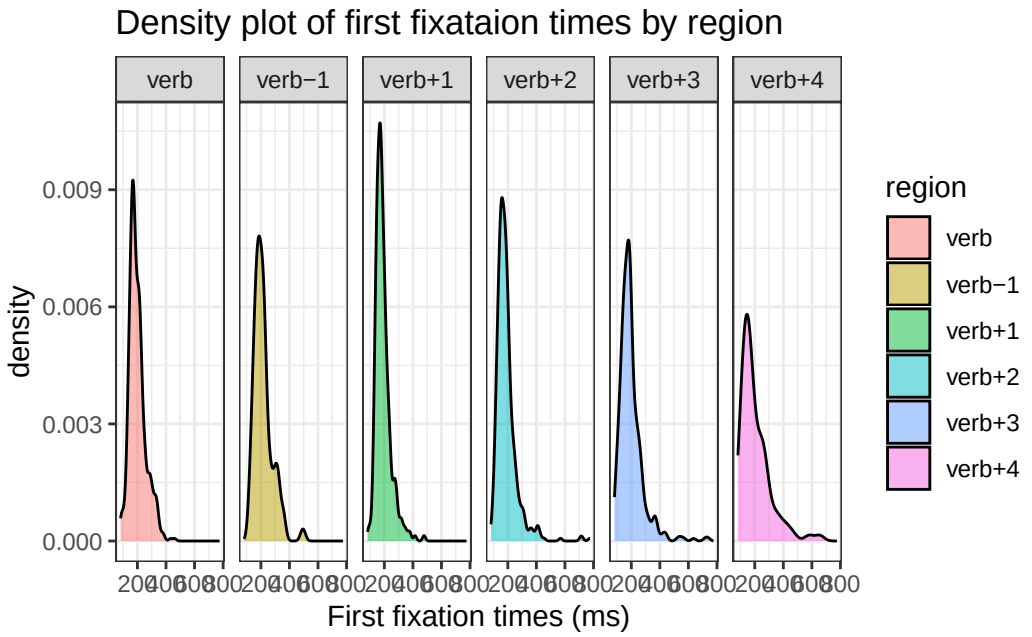


Figure 7: Distribution of first fixation times at the verb region (raw milliseconds).

- how would you describe the density plots of the different regions?

re-ordering factors

- by default, factors will be ordered alphabetically
  - but we don't always want that
  - here, `verb-1` should be before `verb`

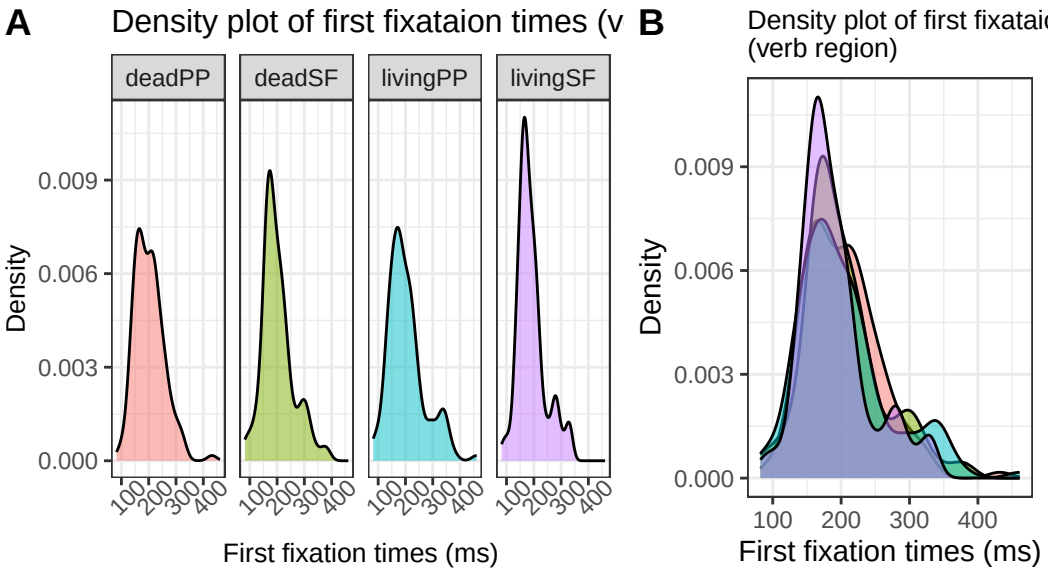
```
df_lifetime <- df_lifetime %>%
  mutate(region = factor(region,
                         levels = c("verb-1","verb","verb+1","verb+2","verb+3","verb+4")))

summary(df_lifetime$region)
```

```
verb-1   verb verb+1 verb+2 verb+3 verb+4
   559    559    559    559    559    182
```

Exercise

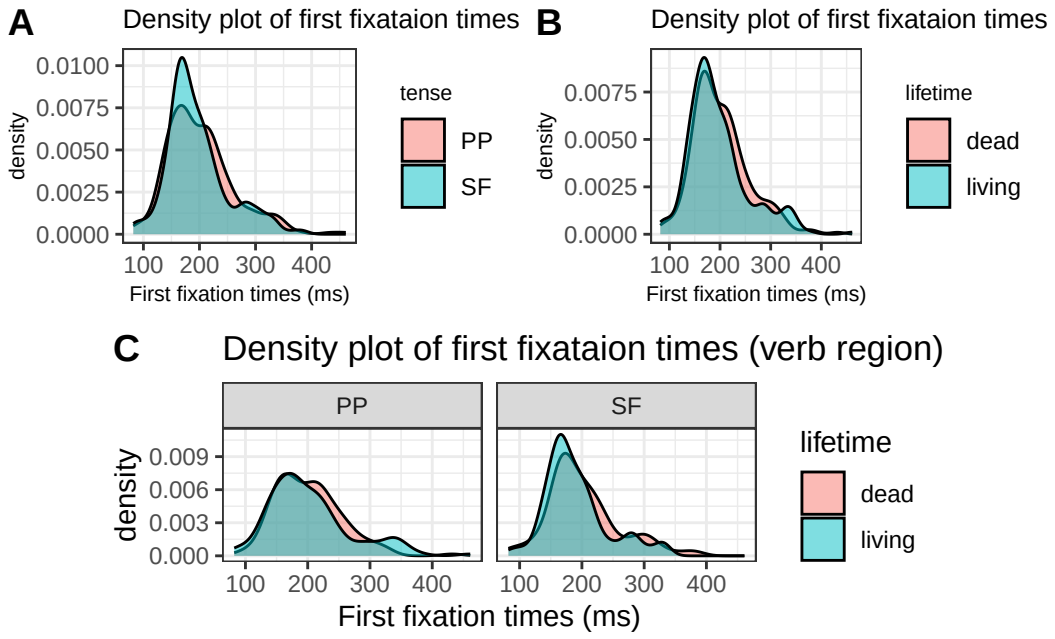
1. create a density plot with the fill colour set to `condition`, but:

- subset the data to only include the verb region
- you can decide if you want to use facets or to have the density curves overlayed
- your plot should look something like A or B:



Extra exercise

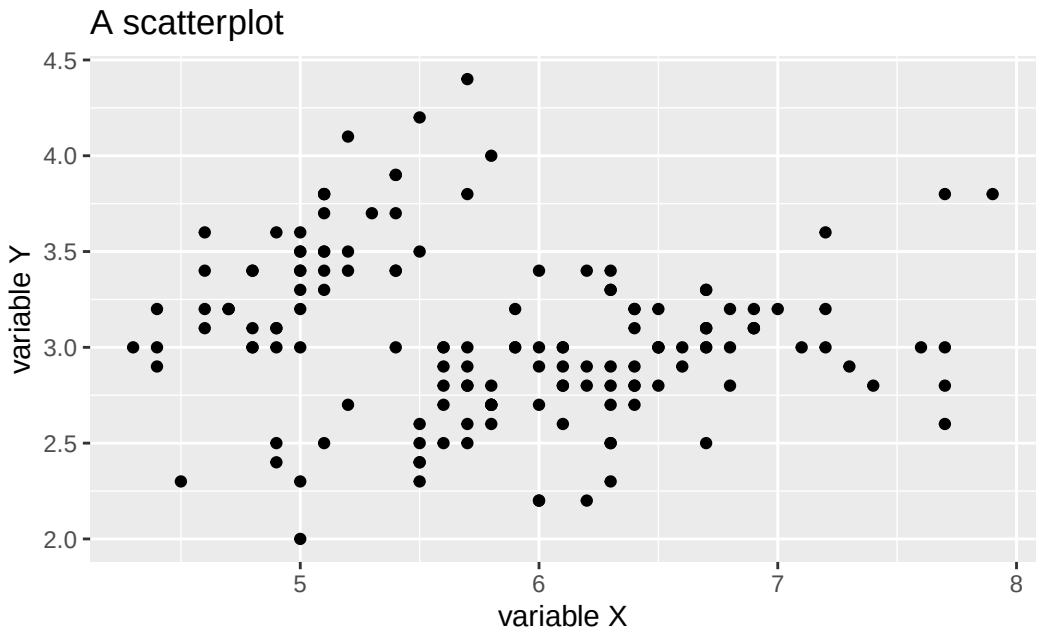
2. Can you produce these plots?



Scatterplots

- histograms and density plots plot a **single variable** along the x-axis
  - in most other plots the dependent (measure) variable is plotted along the y-axis by convention
- scatterplots plot the relationship between **two variables**

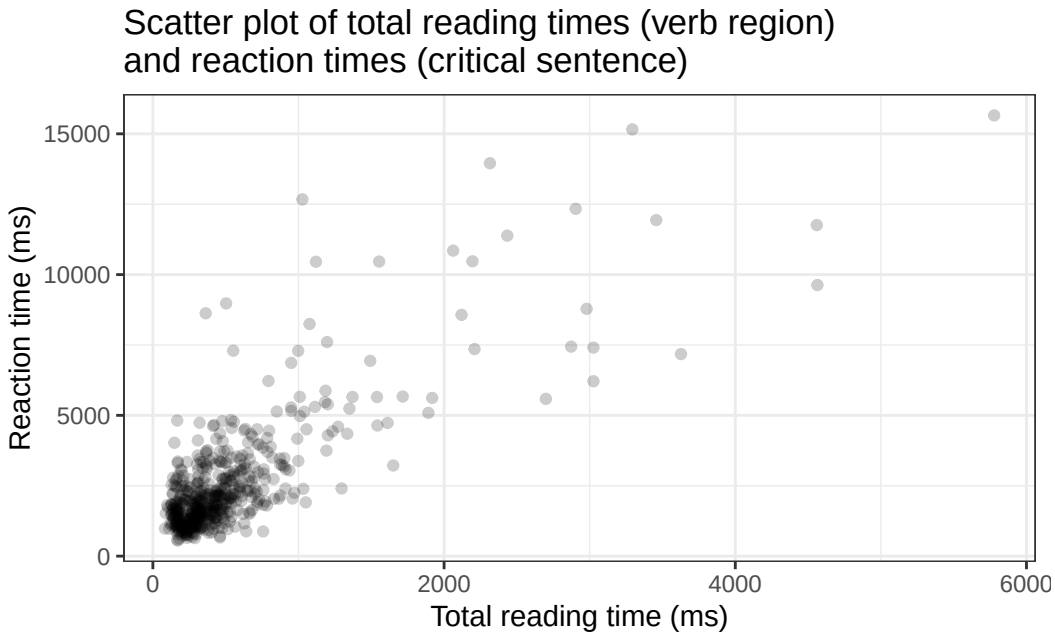
```
iris |>
  ggplot(aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point() +
  labs(title = "A scatterplot",
       x = "variable X",
       y = "variable Y")
```



Scatterplots

- the figure below plots total reading times (verb region) to the verb region (x-axis) and reaction times to the critical sentence (y-axis)
  - what does each point represent?
  - how would you describe the relationship between the two variables?

```
df_lifetime |>
  filter(ff > 0,
         region == "verb") |>
  ggplot(aes(x = tt, y = rt)) +
  labs(title = "Scatter plot of total reading times (verb region)
and reaction times (critical sentence)",
       x = "Total reading time (ms)",
       y = "Reaction time (ms)") +
  geom_point(alpha = .2) +
  theme_bw()
```



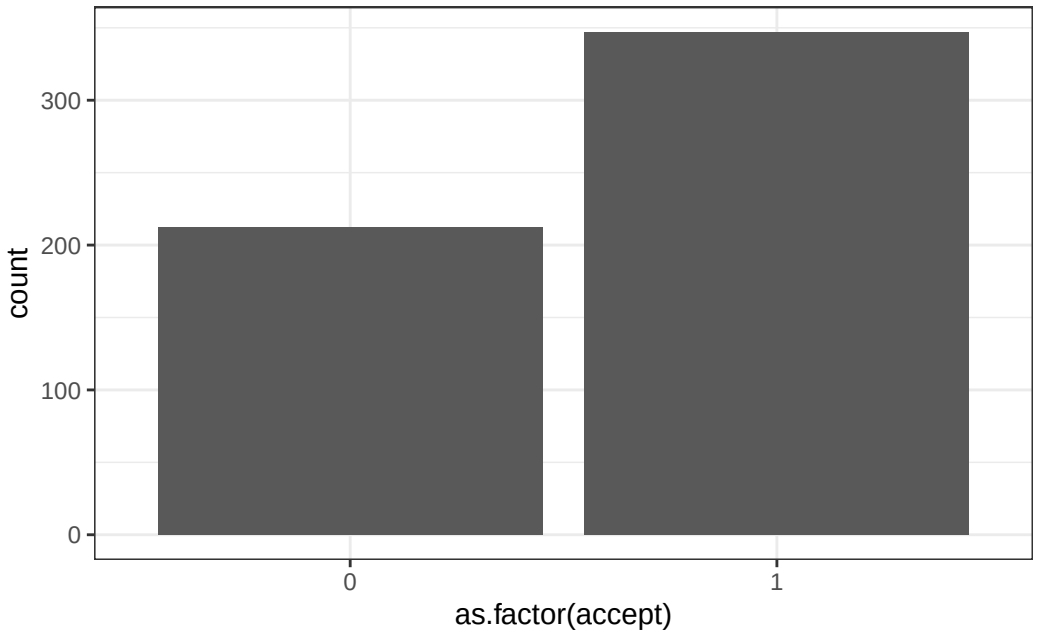
Exercise

1. Generate a scatterplot of total reading times and reaction times, with:
    - colour and shape set to condition
    - tip: these both belong in `aes()`
2. What information does this plot suggest?

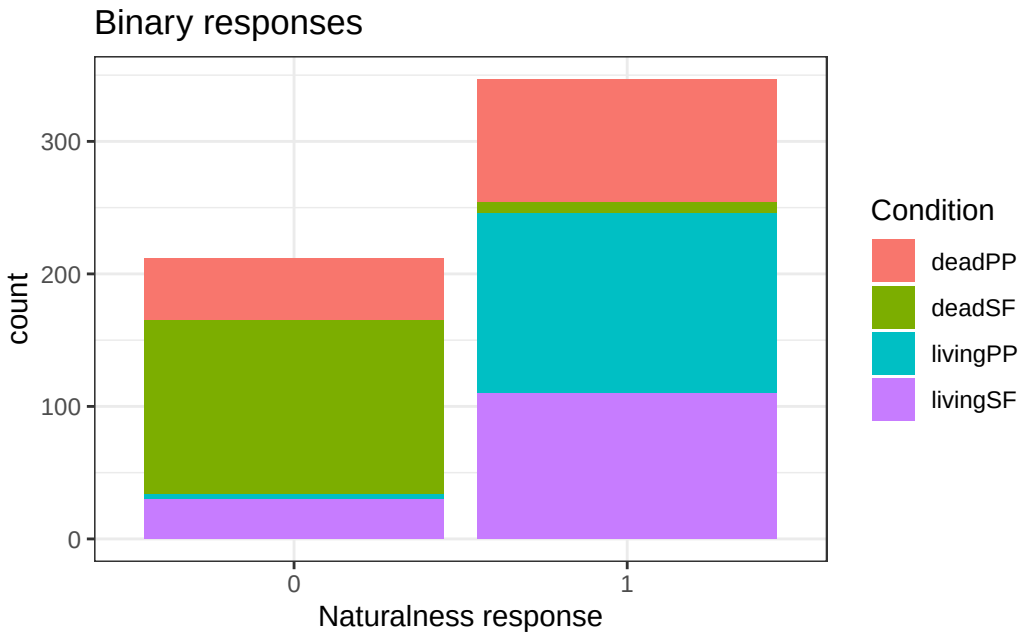
Bar plot

- show the distribution of categorical factor levels
    - i.e., the frequency of observations per level
- be sure to read in `accept` as a factor!

```
df_lifetime |>
  distinct(px,trial,.keep_all=T) |>
  ggplot(aes(x = as.factor(accept) )) +
  geom_bar() +
  theme_bw()
```

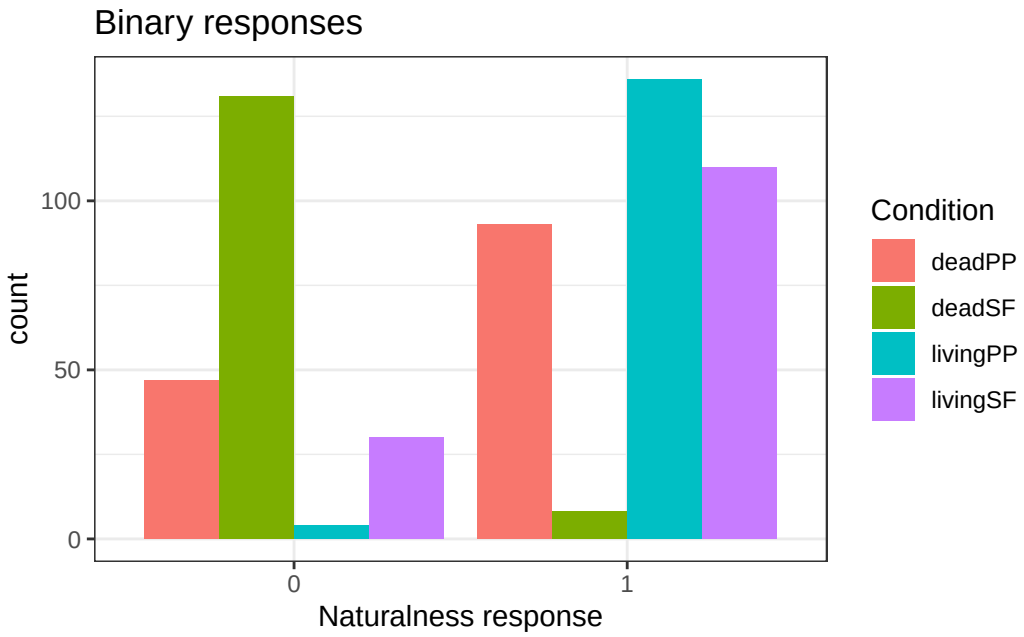



```
df_lifetime |>
  distinct(px,trial,.keep_all=T) |>
  ggplot(aes(x = as.factor(accept), fill = condition)) +
  labs(title = "Binary responses",
       x = "Naturalness response",
       fill = "Condition") +
  geom_bar() +
  theme_bw()
```



Grouped bar plots

```
df_lifetime |>
  distinct(px,trial,.keep_all=T) |>
  ggplot(aes(x = as.factor(accept), fill = condition)) +
  labs(title = "Binary responses",
       x = "Naturalness response",
       fill = "Condition") +
  geom_bar(position = "dodge") +
  theme_bw()
```



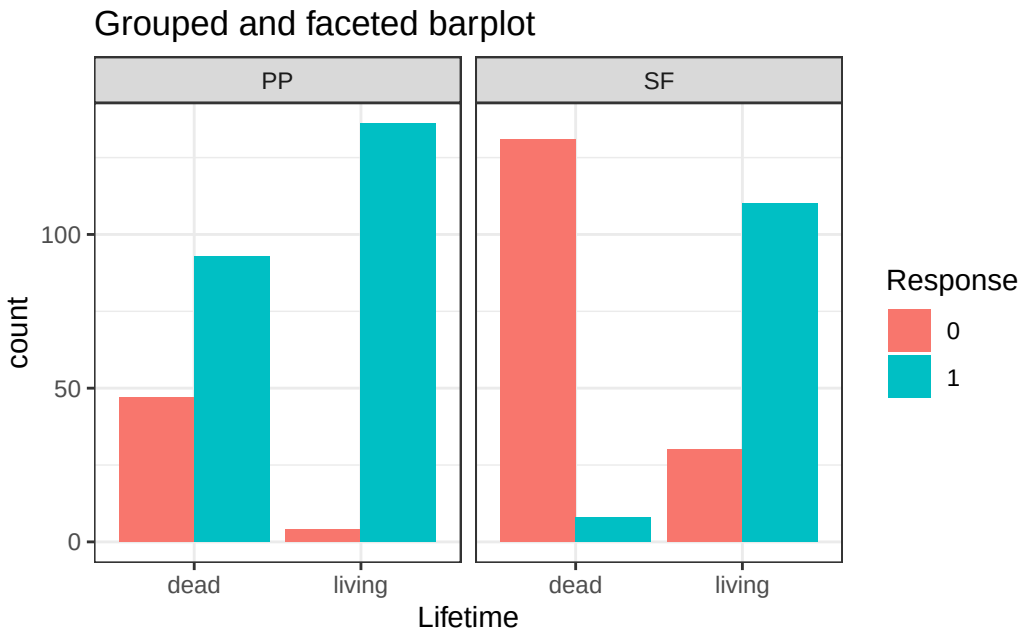
Exercise

1. Generate a grouped bar plot (i.e., `dodge`) with:
    - a facet grid for `tense`
    - plots `lifetime` on the x-axis
    - and fills the bars based on `accept`
    - change the labels accordingly
    - customise as you like

```
df_lifetime |>
  distinct(px,trial,.keep_all=T) |>
  ggplot(aes(x = lifetime, fill = as.factor(accept))) +
  facet_grid(.~tense) +
  labs(title = "Binary responses",
       x = "Lifetime",
       fill = "Response") +
  geom_bar(position = "dodge") +
  theme_bw()
```

Grouped bar plots

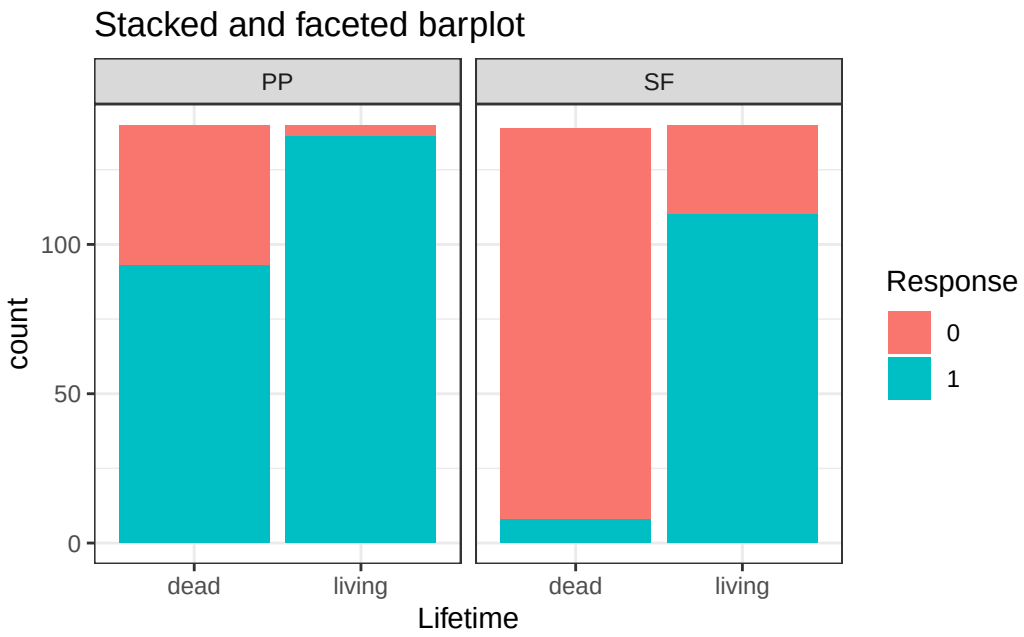
```
df_lifetime |>
  distinct(px,trial,.keep_all=T) |>
  ggplot(aes(x = lifetime, fill = as.factor(accept))) +
  facet_grid(.~tense) +
  labs(title = "Grouped and faceted barplot",
       x = "Lifetime",
       fill = "Response") +
  geom_bar(position = "dodge") +
  theme_bw()
```



Stacked bar plots

```
df_lifetime |>
  distinct(px,trial,.keep_all=T) |>
  ggplot(aes(x = lifetime, fill = as.factor(accept))) +
  facet_grid(.~tense) +
  labs(title = "Stacked and faceted barplot",
       x = "Lifetime",
       fill = "Response") +
```

```
  geom_bar(position = "stack") +
  theme_bw()
```



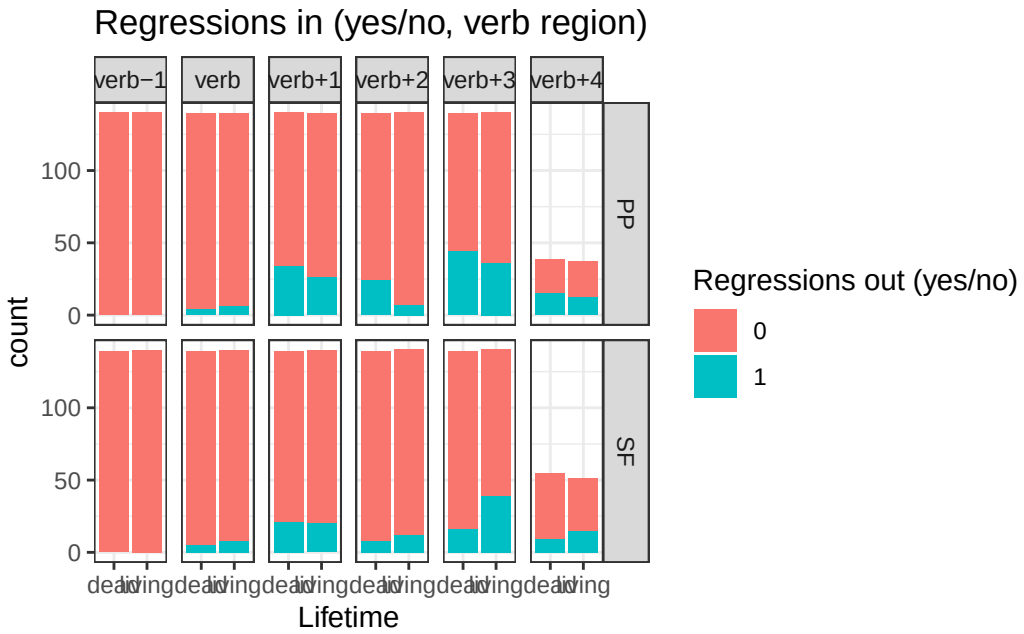
Exercise

1. Choose the barplot you like best for binary data
2. Reproduce that barplot, but with `reg_in` at the `verb1` region

Extra exercise

1. Create another bar plot, but for `reg_out` for all sentence regions
2. Use `facet_grid()`

- to have facets by region (columns) and by tense (in 2 rows)



Summary statistics

- measures of location: mean, median, mode
- measures of spread: (interquartile) range, standard deviation

Boxplots

- boxplots provide information about the distribution of a *continuous* variable
  - but includes information like *median* (dark line) and *quartiles* (box and whiskers)
  - and *outliers* (dots)
- like scatterplots, require x and y variables
  - but one of them needs to be **categorical**

```
iris |>
  ggplot(aes(x = Species, y = Sepal.Length)) +
  labs(title = "A scatterplot",
       x = "Categorical variable",
       y = "Continuous variable") +
  geom_boxplot()
```

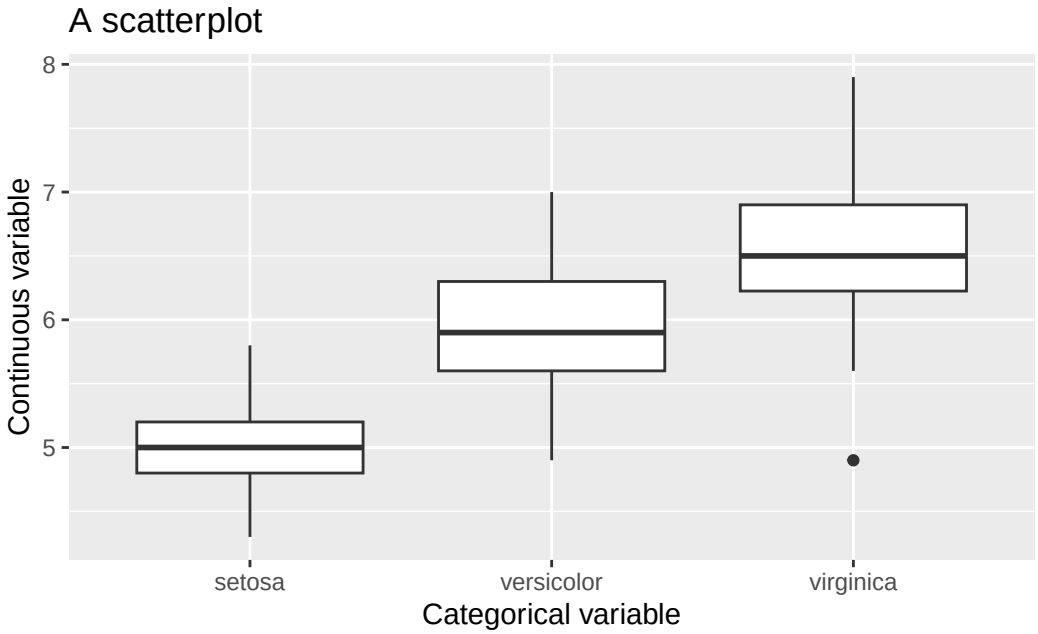


Figure 8: A scatterplot. Median (50th percentile): thick black lines; interquartile range (IQR; 25th and 75th percentile): box limits; minimum (0th percentile) and maximum (100th percentile) excluding outliers: : whiskers; outliers: points

Boxplot explained

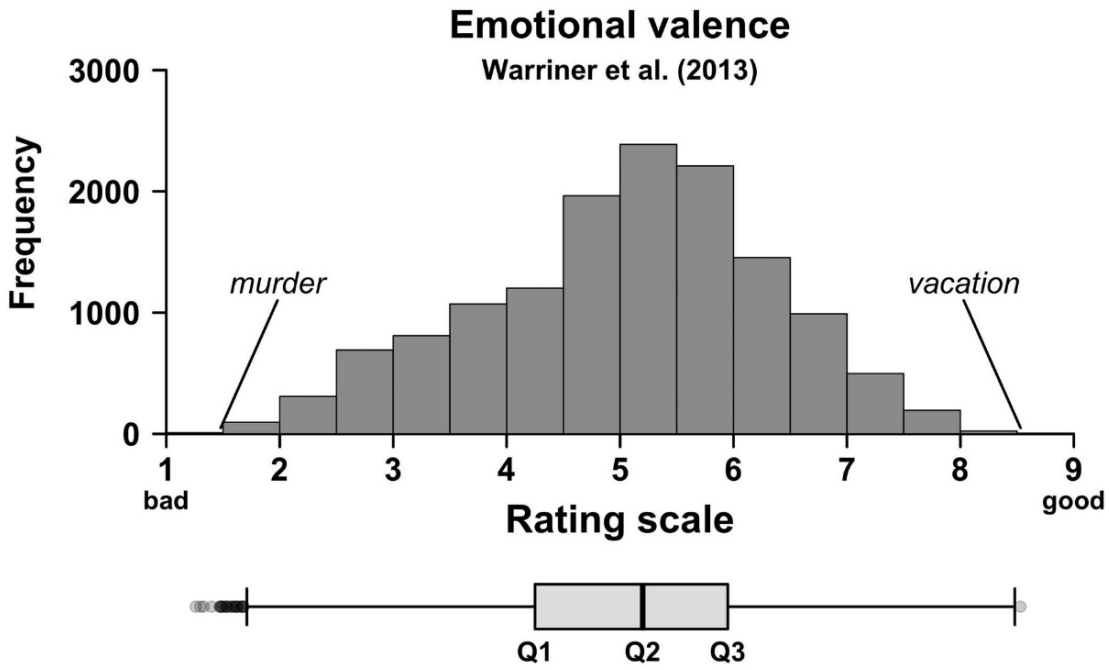


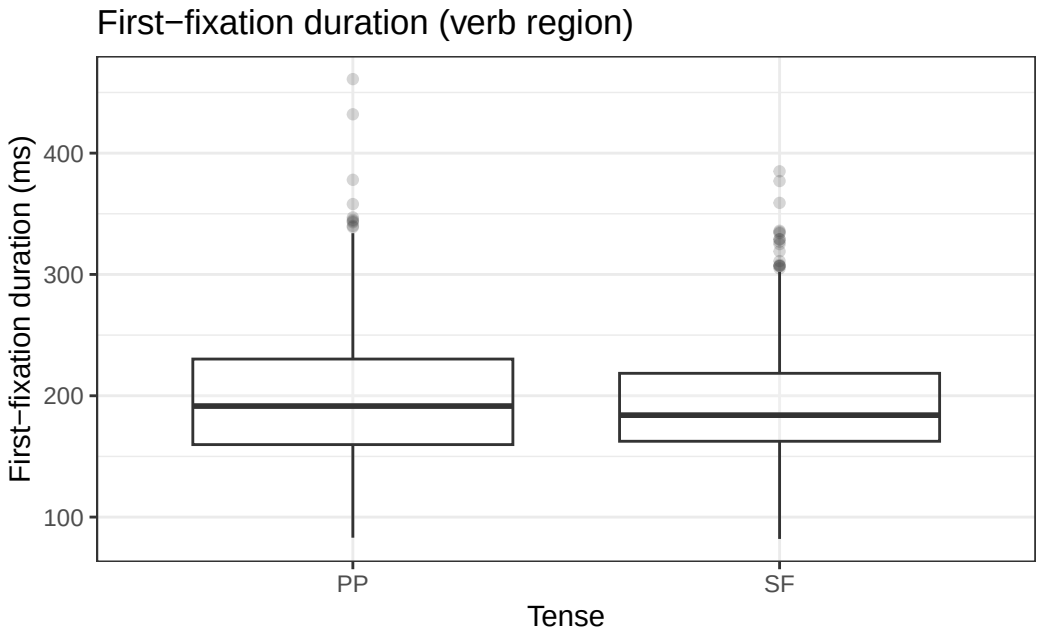
Figure 3.4. A histogram of the emotional valence rating data

Figure 9: Image source: Winter (2019) (all rights reserved)

Boxplots

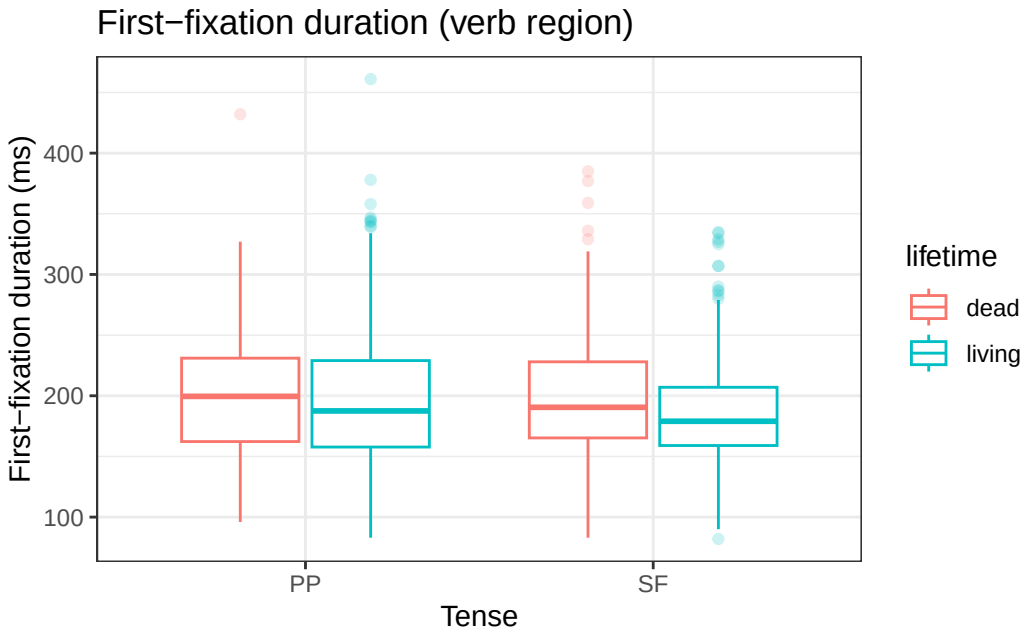
- let's change our scatterplot to a boxplot

```
df_lifetime |>
  filter(ff > 0,
         region == "verb") |>
  ggplot(aes(x = tense, y = ff)) +
  labs(title = "First-fixation duration (verb region)",
       x = "Tense",
       y = "First-fixation duration (ms)") +
  geom_boxplot(alpha = .2) +
  theme_bw()
```



Grouped boxplots

```
df_lifetime |>
  filter(ff > 0,
         region == "verb") |>
  ggplot(aes(x = tense, y = ff, colour = lifetime)) +
  labs(title = "First-fixation duration (verb region)",
       x = "Tense",
       y = "First-fixation duration (ms)") +
  geom_boxplot(alpha = .2) +
  theme_bw()
```



Exercise

1. Create a group boxplot (x = tense, fill = lifetime) for

- first-pass reading time (verb region)
- regression path duration (verb region)
- total reading time (verb region)
- reaction times (use the `distinct()` verb to have a single observation per participant and per trial)

Interaction plots

- common for **factorial** designs, i.e., comparing *categorical* predictors
- there are 2 ways of producing them:
  - with your data frame and `stat_summary()`
  - or with a summary table and ggplot geoms `geom_point()`, `geom_errorbar()`, and `geom_line()`
- we'll need our summary table to plot an interaction plot

condition	lifetime	tense	N	mean.ff	sd	se	ci	lower.ci	upper.ci
deadPP	dead	PP	140	198.9	57.9	4.9	9.7	189.2	208.6
deadSF	dead	SF	139	194.6	67.9	5.8	11.4	183.2	205.9

condition	lifetime	tense	N	mean.ff	sd	se	ci	lower.ci	upper.ci
livingPP	living	PP	140	194.2	77.3	6.5	12.9	181.3	207.1
livingSF	living	SF	140	186.0	57.6	4.9	9.6	176.4	195.6

```
library(patchwork)

df_lifetime |>
  filter(region == "verb") |>
  ggplot(aes(x = lifetime, y = ff,
                      shape = tense,
                      group = tense,
                      color = tense)) +
  labs(title="Interaction plot (`stat_summary()`)",
       x = "Lifetime",
       y = "First fix (ms)",
       shape = "Tense", group = "Tense", color = "Tense", linetype = "Tense") +
  stat_summary(fun = "mean", geom = "point", size = 3, position = position_dodge(0.2)) +
  stat_summary(fun = "mean", geom = "line", position = position_dodge(0.2), aes(linetype=tens
  stat_summary(fun.data = "mean_cl_normal", geom = "errorbar", width = .2
               , position = position_dodge(0.2)) +
  theme_bw() +

summary_ff |>
  ggplot(aes(x = lifetime, y = mean.ff,
                      shape = tense,
                      group = tense,
                      color = tense)) +
  labs(title="Interaction plot (geoms)",
       x = "Lifetime",
       y = "First fix (ms)",
       shape = "Tense", group = "Tense", color = "Tense", linetype = "Tense") +
  geom_point(size = 3,
                position = position_dodge(0.2)) +
  geom_line(aes(linetype=tense), position = position_dodge(0.2)) +
  geom_errorbar(aes(ymin = mean.ff - ci,
                    ymax = mean.ff + ci),
                width = .2,
                position = position_dodge(0.2)) +
  theme_bw() +
  plot_annotation(tag_levels = "A") +
  plot_layout(guides = "collect") &
```

```
  theme(legend.position = "bottom")
```

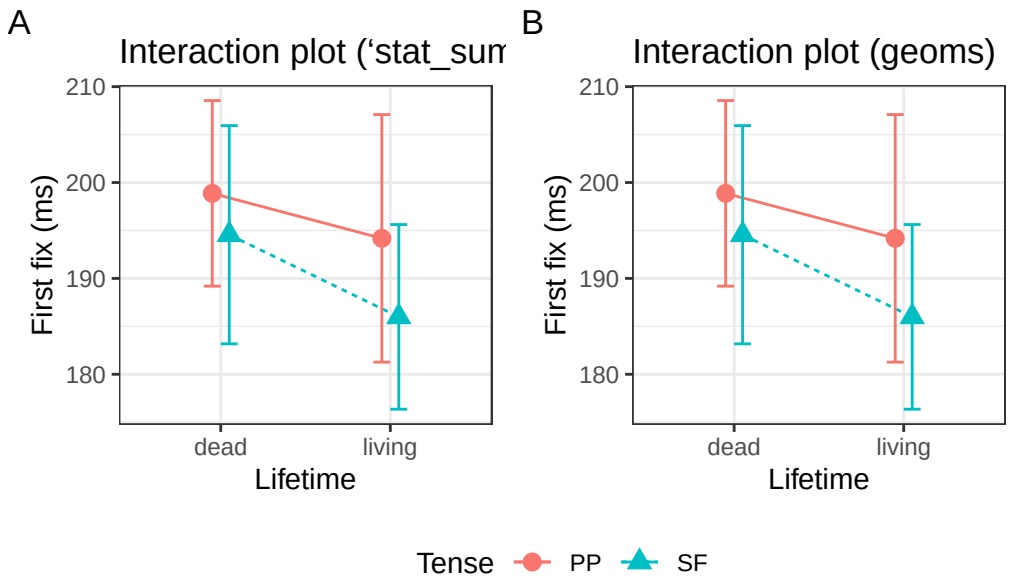


Figure 10: Identical interaction plots produced two ways: feeding a dataset into `ggplot()` and using `stat_summary()` (A) or feeding a summary table into `ggplot()` and using geoms (B)

Question: Binomial data

- binomial data are those with 2 categories, for example
  - present, absent
  - yes, no
- in our dataset, each trial ended with a binary naturalness judgement task
  - how might we plot such data?

Saving our plots

- we can save our plots two ways:
  1. as image files (e.g., JPEG, PNG, SVG, etc.): when writing in Word, LaTeX, etc.
  2. as a single R object (Rds: R data structure): when writing in Rmarkdown/Quarto

ggsave()

- the `ggsave()` function is useful for saving ggplot objects
  - we first have to save one of our figures as an object
  - I usually save ggplot objects with the prefix `fig_` (short for figure)
- make sure you also have a useful place to store these figures
  - e.g., a folder called `figures`

```
fig_lifetime_ff <-
summary_ff |>
  ggplot(aes(x = lifetime, y = mean.ff,
                     shape = tense,
                     group = tense,
                     color = tense)) +
  labs(title="Mean first-fixation times (verb region) with 95% CIs",
       x = "Lifetime",
       y = "First fix (ms)",
       shape = "Tense", group = "Tense", color = "Tense", linetype = "Tense") +
  geom_point(size = 3,
                position = position_dodge(0.2)) +
  geom_line(aes(linetype=tense), position = position_dodge(0.2)) +
  geom_errorbar(aes(ymin = mean.ff - ci,
                    ymax = mean.ff + ci),
                width = .2,
                position = position_dodge(0.2)) +
  theme_bw()
```

```
ggsave(fig_lifetime_ff, filename = here("figures", "fig_lifetime_ff.png"))
```

- `ggsave()` has lots of arguments to control width, height, resolution, etc.
  - to see more, run `?ggsave` in the Console
- you can also save as JPG/JPEG, SVG, even PDF by just changing the filename extension

saveRDS()

- we can also save the figure as R code
  - which means we can control the width, height, resolution, etc. later on when we load it in

  - useful if you'll be writing up your results in R markdown or Quarto

```
saveRDS(fig_lifetime_ff, file = here("figures", "fig_lifetime_ff.rds"))
```

`readRDS()`

- you can't click on the file to view the figure because it's R code
  - you'd need to load the data into R again

```
fig_lifetime_ff <- readRDS(here("figures", "fig_lifetime_ff.rds"))
```

Naming files and saving code

You'll notice I saved the PNG and RDS files using the same name that the I used for the figure in my script. This is an important point: I want to be able to traceback my figures from the code so I can easily track them. It also helps encourage informative object and file names.
Of course, saving the code used to save the files in our scripts is also useful because we can easily adjust the saved files (e.g., change figure width or height)

References

Nordmann, E., & DeBruine, L. (2022). *Applied data skills*. Zenodo. https://doi.org/10.5281/zenodo.6365078

Nordmann, E., McAleer, P., Toivo, W., Paterson, H., & DeBruine, L. M. (2022). Data Visualization Using R for Researchers Who Do Not Use R. *Advances in Methods and Practices in Psychological Science*, *5*(2), 251524592210746. https://doi.org/10.1177/25152459221074654

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science* (2nd ed.).

Winter, B. (2019). Statistics for Linguists: An Introduction Using R. In *Statistics for Linguists: An Introduction Using R*. Routledge. https://doi.org/10.4324/9781315165547