Regression for Linguists

WiSe23/24

Daniela Palleschi

Thu, Sep 07, 2023

Table of contents

Course overview

This course fast tracks through different types of regression most relevant to linguistic research. These materials are by no means exhaustive, and should be supplemented by reading textbook length treatments. The majority of my materials lean heavily on Winter (2019), which I highly recommend. I also took inspiration from Sonderegger (2023), which came out this year and I haven't fully explored yet. So far, it looks like a very thorough textbook that I would also recommend you check out.

Before you begin the course, I would like to paraphrase something Prof. Shravan Vasishth said in the opening remarks for the annual summer school for Statistcal Methods for Linguistics and Psychology back in 2020 which has stuck with me: *get comfortable with partial knowledge*. We are not trained statisticians, and likely never will be (Vasishth himself is a certified statistician, in addition to professor of psycholinguistics). So get comfortable with partial understanding of the math behind these models, and focus on their application and interpretation.

Materials

This website is a work-in-progress. Materials will be updated/brushed up throughout the semester, with the binding course materials available on the course Moodle for those enrolled in the winter semester 2023/24.

This website was created to be viewed in HTML format. The accompanying (PDF) book version can be accessed by clicking on the PDF icon at the top right, but is not optimally formatted. Tables formatted for HTML output are particularly oddly formatted in PDF, as is the order of printed elements in relation to their accompanying text. For this reason, I strongly encourse to follow the web book.

Part I

Overview

Syllabus

Meeting	Lecture	Topic	Vorbereitung
2023-10-10	1	Equation of a line	Winter (2019): Ch. 1-3
2023-10-11	2	Linear regression	Winter (2019): Ch. 4 Winter (2013)
2023-10-12	3	Continuous predictors	Winter (2019): Ch. 5 Winter (2013)
2023-10-10	4	Multiple linear regression	Winter (2019): Ch. 6 Winter (2013)
2023-10-11	5	Categorical predictors	Winter (2019): Ch. 7 Winter (2013)
2023-10-12	6	Model assumptions	
2023-10-10	7	Logistic regression	Winter (2019): Ch. 12
2023-10-11	8	Log odds, logits, and odds ratio	
2023-10-12	9	Foundational Ideas	Vasishth & Nicenboim (2016)
2024-01-12	10	Linear mixed models	Winter (2019): Ch. 14 Winter & Grice (2021); unt
2024-01-12	11	Linear mixed models	
2024-01-26	12		
2024-01-26	13		
2024-02-09	14		
2024-02-09	15		

Resources and Set-up

Resources

This course is mainly based on Winter (2019), which is an excellent introduction into regression for linguists. For even more introductory tutorials, I recommend going through Winter (2013) and Winter (2014) For a more intermediate textbook, I'd recommend Sonderegger (n.d.).

If you're interested in the foundational writings on the topic of (frequentist) linear mixed models in (psycho)linguistic research, I'd recommend reading Baayen (2008); Baayen et al. (2008);Barr et al. (2013); Bates et al. (2015); Jaeger (2008); Matuschek et al. (2017); Vasishth (2022); Vasishth & Nicenboim (2016).

Assumptions about you

For this course, I assume that you are familiar with more classical statistical tests, such as the t-test, Chi-square test, etc. I also assume you are familiar with measures of central tendency (mean, median, mode) measures dispersion/spread (standard deviation), and with the concept of a normal distribution. Lacking this knowledge will not impeded your progress in the course, but is an important foundation on which we'll be building. We can review these concepts in-class as needed.

Software

- R: a statistical programming language (the underlying language)
- RStudio: an program that facilitates working with R; our preferred IDE integrated development environment
- LaTeX: a typesetting system that generates documents in PDF format
- why R?
  - R and RStudio are open-source and free software
  - they are widely used in science and business

Install R

- we need the free and open source statistical software R to analyze our data
- download and install R: https://www.r-project.org

Install RStudio

- we need RStudio to work with R more easily
- Download and install RStudio: https://rstudio.com
- it can be helpful to keep English as language in RStudio
  - we will find more helpful information if we search error messages in English on the internet
- If you have problems installing R or RStudio, check out this help page (in German): http://methods-berlin.com/wp-content/uploads/Installation.html

Install LaTeX

- we will not work with LaTeX directly, but it is needed in the background
- Download and install LaTeX: https://www.latex-project.org/get/

resources

- many aspects of this course are inspired by (**nordmann_applied_2022?**) and (**wickham_r_nodate?**)
  - both freely available online (in English)
- for German-language resources, visit the website of Methodengruppe Berlin

Troubleshooting (EN: Troubleshooting)

- Error messages are very common in programming, at all levels.
- How to find solutions for these error messages is an art in itself
- Google is your friend! If possible, google in English to get more information

References

Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics using R*.

Baayen, R. H., Davidson, D. J., & Bates, D. M. (2008). Mixed-effects modeling with crossed random effects for subjects and items. *Journal of Memory and Language*, *59*(4), 390–412. https://doi.org/10.1016/j.jml.2007.12.005

Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing linguistic data: A practical introduction to statistics*. https://CRAN.R-project.org/package=languageR

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language*, *68*(3), 255–278. https://doi.org/10.1016/j.jml.2012.11.001

Bates, D., Kliegl, R., Vasishth, S., & Baayen, H. (2015). Parsimonious Mixed Models. *arXiv Preprint*, 1–27. https://doi.org/10.48550/arXiv.1506.04967

Biondo, N., Soilemezidi, M., & Mancini, S. (2022). Yesterday is history, tomorrow is a mystery: An eye-tracking investigation of the processing of past and future time reference during sentence reading. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *48*(7), 1001–1018. https://doi.org/10.1037/xlm0001053

Jaeger, T. F. (2008). Categorical data analysis: Away from ANOVAs (transformation or not) and towards logit mixed models. *Journal of Memory and Language*, *59*(4), 434–446. https://doi.org/10.1016/j.jml.2007.11.007

Lüdecke, D., Ben-Shachar, M. S., Patil, I., Waggoner, P., & Makowski, D. (2021). performance: An R package for assessment, comparison and testing of statistical models. *Journal of Open Source Software*, *6*(60), 3139. https://doi.org/10.21105/joss.03139

Matuschek, H., Kliegl, R., Vasishth, S., Baayen, H., & Bates, D. (2017). Balancing Type I error and power in linear mixed models. *Journal of Memory and Language*, *94*, 305–315. https://doi.org/10.1016/j.jml.2017.01.001

Sonderegger, M. (n.d.). *Regression Modeling for Linguistic Data*.

Sonderegger, M. (2023). *Regression Modeling for Linguistic Data*.

Vasishth, S. (2022). *Some right ways to analyze (psycho)linguistic data* [Preprint]. PsyArXiv. https://doi.org/10.31234/osf.io/y54va

Vasishth, S., & Nicenboim, B. (2016). Statistical methods for linguistic research: Foundational Ideas—Part I. *Language and Linguistics Compass*, *10*(11), 591–613. https://doi.org/10.1111/lnc3.12207

Winter, B. (2013). *Linear models and linear mixed effects models in R: Tutorial 1*.

Winter, B. (2014). *A very basic tutorial for performing linear mixed effects analyses (Tutorial 2)*.

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

Part II

Day 1: Simple linear regression

1 Understanding straight lines

Regression for Linguists

```
# suppress scientific notation
options(scipen=999)
```

```
# load libraries
pacman::p_load(tidyverse,
               broom,
               patchwork,
               knitr,
               kableExtra,
               gt,
               googlesheets4)

# tell googlesheets4 we don't want private
gs4_deauth()
```

Learning Objectives

Today we will learn…

- the equation of a line
- about intercepts, slopes, and residuals

Resources

This lecture is based on the readings for today's session: Winter (2013) and Winter (2019) (Ch. 3), and to a lesser extent (**debruine_understanding_2021?**); Winter (2014).

1.1 When to model your data

By the time we get to the point of wanting to model our data, we should have a pretty good idea of how our data look. We achieve this through running an exploratory data analysis (EDA), which consists of visualising your data and determining outliers (a question for another day: what *is* an outlier?), generating summary (i.e., descriptive) statistics, and just overall getting to know your data, without making any claims beyond your data.

However, an understanding of the data design and collection procedure is incredibly important and is necessary in order to appropriately fit a model to our data. In fact, planning out your analyses when designing your experiment is highly recommended in order to ensure your data will have the appropriate structure and that the assumptions made by your chosen analyses are taken into consideration before data collection.

The next step after conducting an EDA is to *model* your data, i.e., run **inferential statistics**, this is where we try to generalise beyond our data.

1.1.1 Statistical tests versus models

Many statistical courses and textbooks still put undue emphasis on classical statistical tests. However, these common statistical tests are simplified linear models, without the added benefits of linear models. In essence, statistical tests tell us something about our data, whereas statistical *models* can be used to make predictions about hypothetical future observations.

1.2 (Linear) Regression

Data exploration gives us an idea about what our data look like, but if we want to be able to make predictions about hypothetical observations, i.e., to *predict* values of our DV based on one (or more) IV(s), we need to fit a model to our data. This model can then *predict* values of our DV based on one (or more) IV(s), i.e., *predicting* an outcome variable (dependent variable, DV) from one or more predictors (independent variable, IV). Because we're making predictions, we need to take into account the variability (i.e., *error*) in our data.

1.2.1 Types of regression

1.3 Straight lines

- *linear regression* summarises the data with a straight line
  - we *model* our data as/fit our data to a straight line

regression type	predictor	outcome
simple regression	Single predictor	continuous (numerical)
multiple regression	multiple predictor	continuous (numerical)
hierarchical/linear mixed models/linear mixed effect models	include random effect	continuous (numerical)
generalised linear (mixed) models: logistic regression	as above	binary/binomial data
generalised linear (mixed) models: poisson regression	as above	count data

- *straight lines* can be defined by
  - Intercept (b_0)
    - value of Y when $X = 0$
  - Slope (b_1)
    - gradient (slope) of the regression line
    - direction/strength of relationship between x and y
    - regression coefficient for the predictor
- so we need to define an intercept and a slope

1.3.1 A line = intercept and slope

- a line is defined by its intercept and slope
  - in a regression model, these two are called `coefficients`

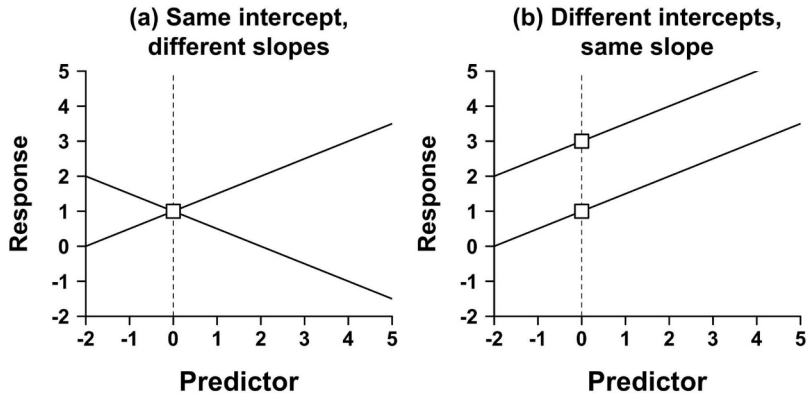


Figure 4.2. (a) Two lines with positive and negative slopes that go through the same intercept; (b) two lines with the same positive slope that have different intercepts

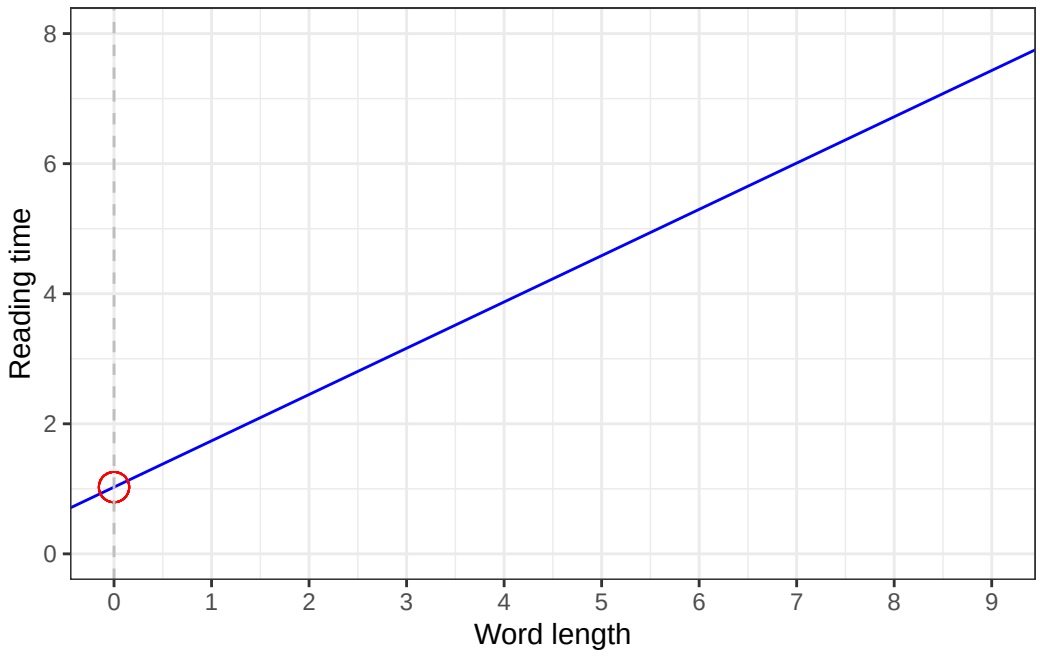
Figure 1.1: Image source: Winter (2019) (all rights reserved)

Equation of a line

$$y = mx + c \tag{1.1}$$
$$Y_i = (b_0 + b_1 X_i) \tag{1.2}$$
$$outcome_i = (model) \tag{1.3}$$
$$y_i = (intercept + slope * x_i) \tag{1.4}$$

1.3.2 Intercept (b_0)

- the value of y when $x = 0$



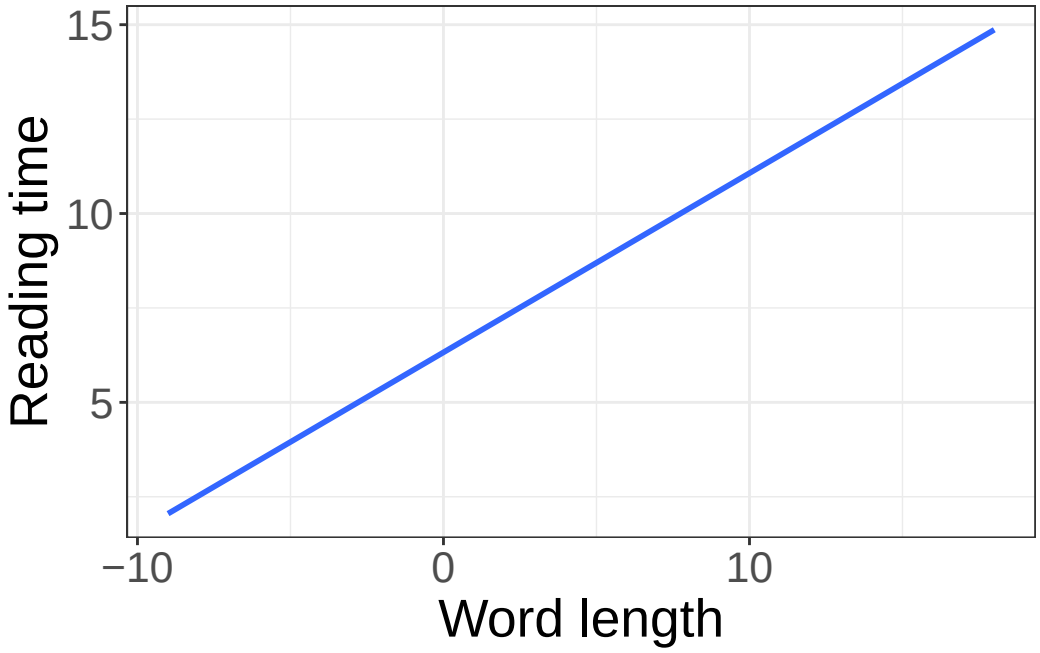
1.3.3 Slopes (b_1)

A slope describes a *change in* x (Δx) over a *change in* y (Δy), where Δ (the Greek letter *delta*) can be read as 'difference' (because it also starts with a 'd'…). So a slope's value equals the difference in x for a difference of 1 unit in y. Positive slopes indicate that as x increases, y increases. A negative slope value indicates that as x increases, y decreases (or vice versa). A slope of 0 indicates there is no change in y as a function of x, or: there is no change in y when the value of x changes.

$$slope = \frac{\Delta x}{\Delta y} \tag{1.5}$$

This relationship between x and y is sometimes referred to as "rise over run": how do you 'rise' in y for a given 'run' in x? For example, if we were to measure children's heights and ages, we would expect to find an increase in height for every increase in age. Or, for a linguistic example, we would expect to find longer whole-sentence *reading times* (a measure variable) for longer texts: if a sentence has 9 words (*I find straight lines to be really interesting and fun.*), we would expect longer reading times than a sentence with 3 words (*I love lines.*).

- what is the intercept of this line?
- what is the slope of this line?



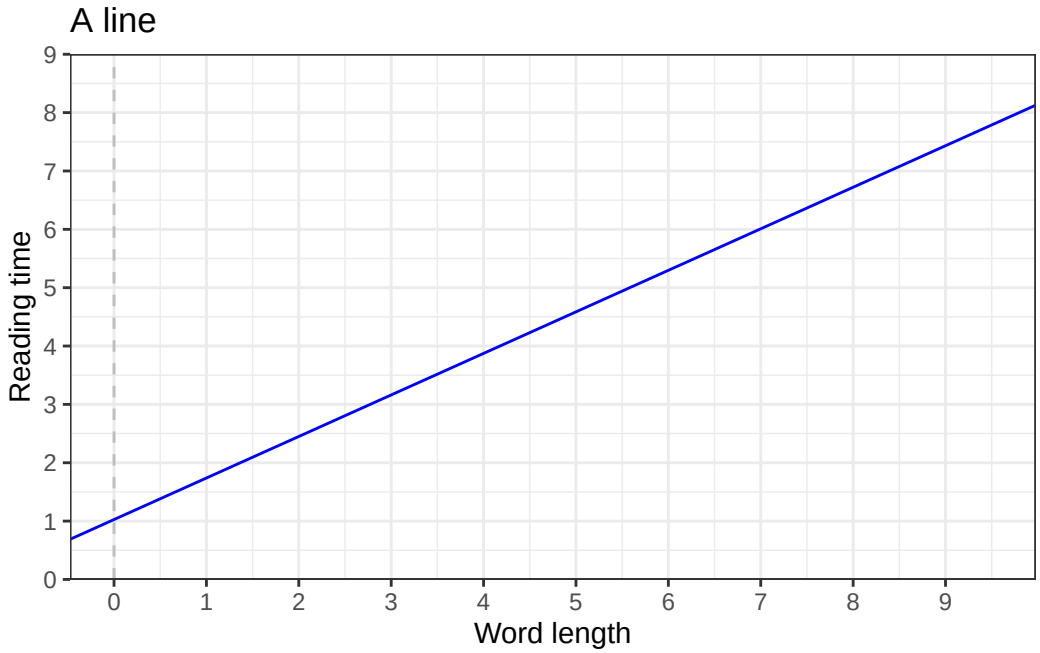
1.4 Error and residuals

- *fixed effects* (IV/predictors): things we can understand/measure
- *error* (random effects): things we cannot understand/measure
  - in biology, social sciences (and linguistic research), there will always sources of random error that we cannot account for
  - random error is less an issue in e.g., physics (e.g., measuring gravitational pull)

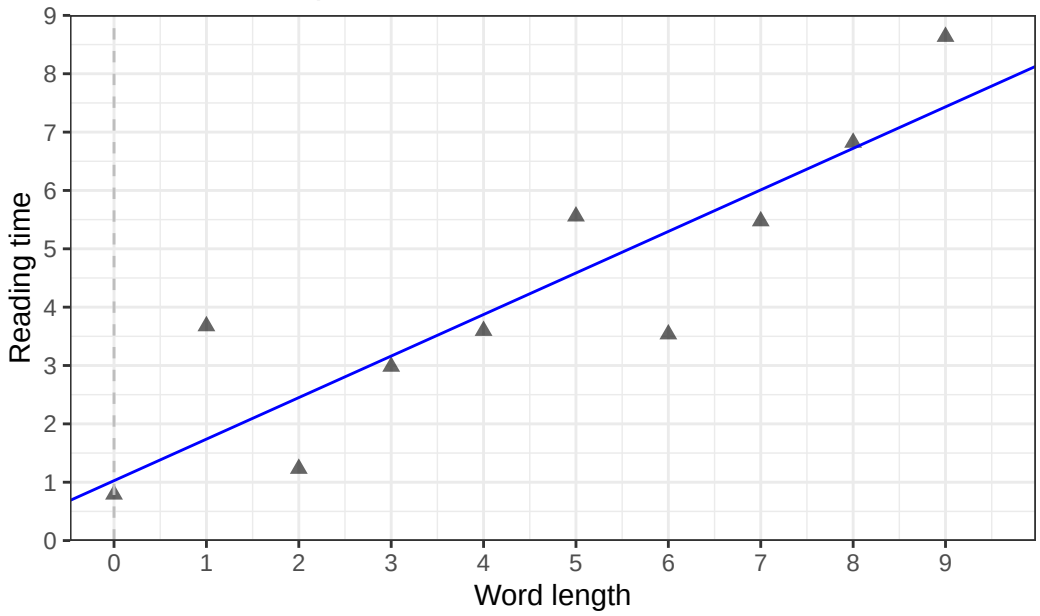
- *residuals*: the difference (vertical difference) between **observed data** and the **fitted values** (predicted values)

Equation of a line

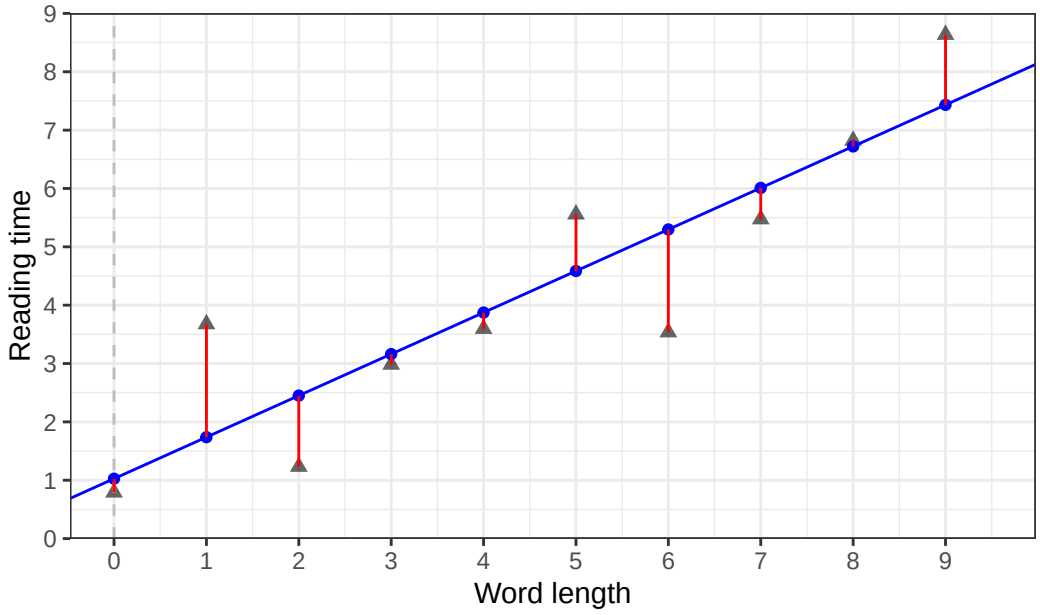
$$y = mx + c \tag{1.6}$$
$$Y_i = (b_0 + b_1 X_i) + \epsilon_i \tag{1.7}$$
$$outcome_i = (model) + error_i \tag{1.8}$$
$$y_i = (intercept + slope * x_i) + error_i \tag{1.9}$$



A line with data points



A line with data points and regression line



1.4.1 Method of least squares

- so how is any given line chosen to fit any given data?
- the ***method of least squares***
  - take a given line, and square all the residuals (i.e., $residual^2$)
  - the line with the lowest ***sum of squares*** is the line with the best fit to the given data
  - why do we square the residuals before summing them up?
    - so all values are positive (i.e., so that negative values don't cancel out positive values)
- this is how we find the ***line of best fit***
  - R fits many lines to find the one with the best fit

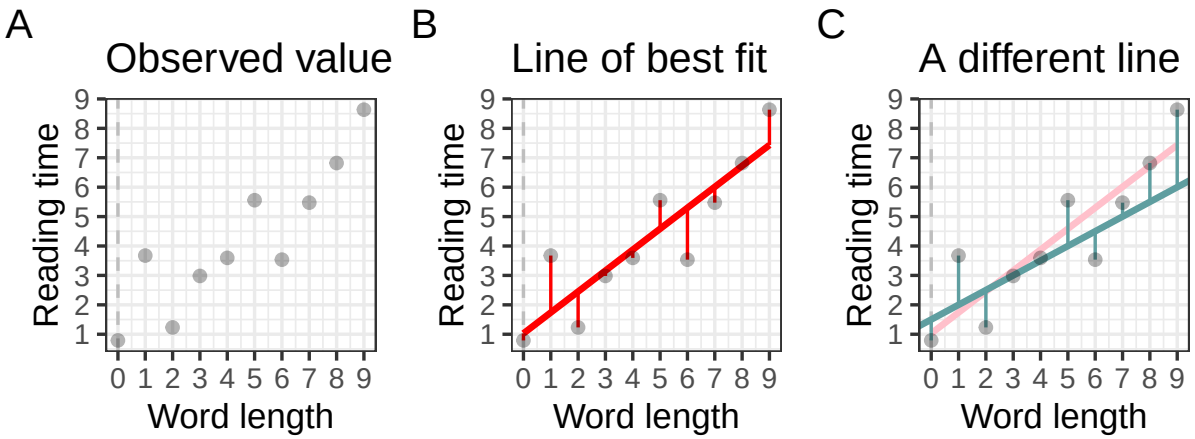


Figure 1.2: Observed values (A), Residuals for line of best fit (B), A line of worse fit with larger residuals (C)

Learning Objectives

Today we learned...

Important terms

Term	Definition
Intercept	Value of y for x=0
Slope	a change in x over a change in y; regression coefficient for the predictor. Positive as x incre
residuals/error	difference between observed data and the fitted values
interaction term	used to describe how effects of one predictor may be influenced by changes in another pred

1.5 Tasks

1.5.1 Task 1: pen-and-paper

You will receive a piece of paper with several grids on it. Follow the instructions, which include drawing some lines. If you aren't in-class, this is the paper we are using:

1.5.2 Task 2: simulating data

All of the figures we just saw (except Figure 1.1, which is from Winter (2019)) were generated in R. Simulating data and plotting is a great way to understand concepts, or even to map out our hypotheses. Let's use R for the first time to try to simulate some data in order to plot lines. Our goal will be to produce a line that has the following:

- intercept = 4.5
- slope = 3

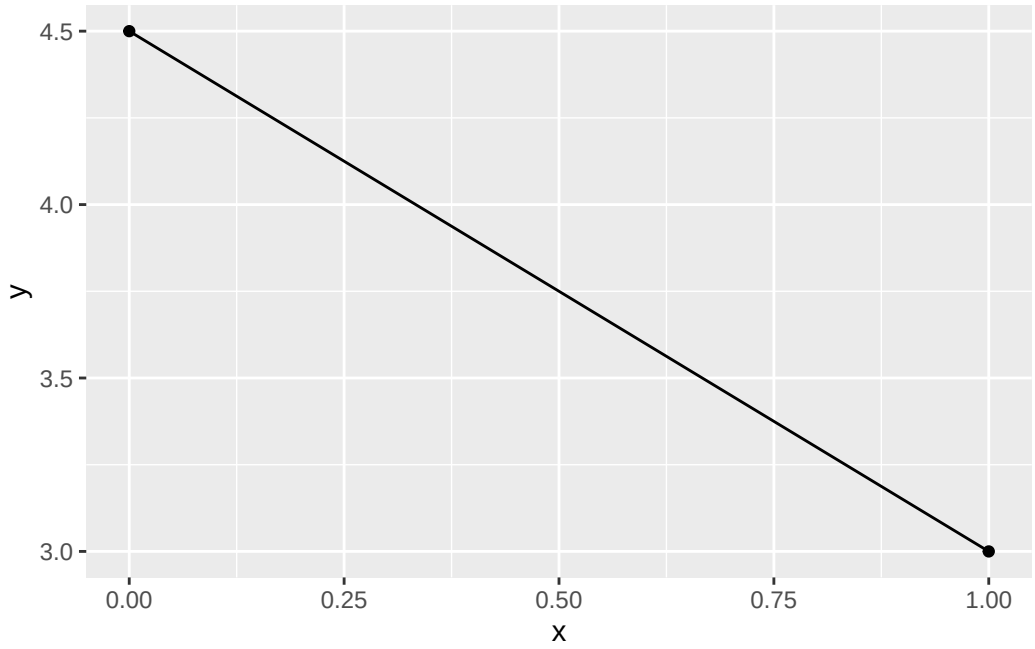
1.5.2.1 Planning

First, think about what steps will be required to create such plots. Can you come up with a workflow plan (without peaking at the next tasks)?

1.5.2.2 Producing our line

```
x <- c(0,1)
y <- c(4.5,3)
data <- cbind(x,y) |> as.data.frame()
ggplot(data = data) +
  aes(x = x, y = y) +
```

```
  geom_line() +
  geom_point()
```



Literaturverzeichnis

Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics using R*.

Baayen, R. H., Davidson, D. J., & Bates, D. M. (2008). Mixed-effects modeling with crossed random effects for subjects and items. *Journal of Memory and Language*, *59*(4), 390–412. https://doi.org/10.1016/j.jml.2007.12.005

Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing linguistic data: A practical introduction to statistics*. https://CRAN.R-project.org/package=languageR

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language*, *68*(3), 255–278. https://doi.org/10.1016/j.jml.2012.11.001

Bates, D., Kliegl, R., Vasishth, S., & Baayen, H. (2015). Parsimonious Mixed Models. *arXiv Preprint*, 1–27. https://doi.org/10.48550/arXiv.1506.04967

Biondo, N., Soilemezidi, M., & Mancini, S. (2022). Yesterday is history, tomorrow is a mystery: An eye-tracking investigation of the processing of past and future time reference during sentence reading. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *48*(7), 1001–1018. https://doi.org/10.1037/xlm0001053

Jaeger, T. F. (2008). Categorical data analysis: Away from ANOVAs (transformation or

not) and towards logit mixed models. *Journal of Memory and Language*, *59*(4), 434–446. https://doi.org/10.1016/j.jml.2007.11.007

Lüdecke, D., Ben-Shachar, M. S., Patil, I., Waggoner, P., & Makowski, D. (2021). performance: An R package for assessment, comparison and testing of statistical models. *Journal of Open Source Software*, *6*(60), 3139. https://doi.org/10.21105/joss.03139

Matuschek, H., Kliegl, R., Vasishth, S., Baayen, H., & Bates, D. (2017). Balancing Type I error and power in linear mixed models. *Journal of Memory and Language*, *94*, 305–315. https://doi.org/10.1016/j.jml.2017.01.001

Sonderegger, M. (n.d.). *Regression Modeling for Linguistic Data*.

Sonderegger, M. (2023). *Regression Modeling for Linguistic Data*.

Vasishth, S. (2022). *Some right ways to analyze (psycho)linguistic data* [Preprint]. PsyArXiv. https://doi.org/10.31234/osf.io/y54va

Vasishth, S., & Nicenboim, B. (2016). Statistical methods for linguistic research: Foundational Ideas—Part I. *Language and Linguistics Compass*, *10*(11), 591–613. https://doi.org/10.1111/lnc3.12207

Winter, B. (2013). *Linear models and linear mixed effects models in R: Tutorial 1*.

Winter, B. (2014). *A very basic tutorial for performing linear mixed effects analyses (Tutorial 2)*.

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

2 Simple linear regression

Regression for Linguists

Learning Objectives

Today we will learn...

- how to fit a simple linear model with the `lm()` function
- how to interpret our model output

Set-up environment

Make sure you always start with a clean R Environment (`Session > Restart R`). This means you should have no objects stored in your Environment, and no packages loaded. To ensure this, you can go to the `Session` tab (up where you'll find `File`, `Help`, etc.), and select `Restart R`. You can also use the keyboard shortcut `Cmd/Ctrl+Shift+0` (that's a zero, not an 'oh').

In addition, I often prefer to run `options(scipen=999)` in order to supress scientific notation, which writes very large or very small numbers in an unintuitive way. For example, 0.000005 is written 5e-06 in scientific notation.

```
# suppress scientific notation
options(scipen=999)
```

We'll also need to load in our required packages. Hopefully you've already install the required packages (if not, go to **?@sec-software**).

```
# load libraries
pacman::p_load(
               tidyverse,
               here,
```

```
                broom,
                lme4,
                janitor,
                languageR)
```

2.1 Simple linear model: RT ~ frequency

Recall that $y \sim x$ can be read as "y as a function of x", or "y predicted by x". Following Winter (2019), we will first model some word frequency data. In this experiment, Our first model is given in equation 2.1:

$$RT \sim frequency \tag{2.1}$$

Let's load our data using the `read_csv()` function from `readr`. I also use the `clean_names()` function from the `janitor` package, which tidies up variable names (e.g., no spaces, all lower case).

```
# load ELP_frequency.csv
df_freq <- read_csv(here("data", "ELP_frequency.csv")) |>
  clean_names()
```

2.1.1 Mini-EDA

Let's explore the data a little bit, which is what we would normally do before fitting any models. First, let's see how the data is structured.

```
# print head of df_freq
head(df_freq)
```

```
# A tibble: 6 x 3
  word      freq    rt
  <chr>    <dbl> <dbl>
1 thing    55522  622.
2 life     40629  520.
3 door     14895  507.
4 angel     3992  637.
5 beer      3850  587.
6 disgrace   409  705
```

Looks like there are only 3 columns: `word`, `freq`, and `rt`. We can assume that they correspond to the word, its frequency, and the reaction time, respectively. We can also see in our global environment that there are 12 observations, meaning 12 rows.

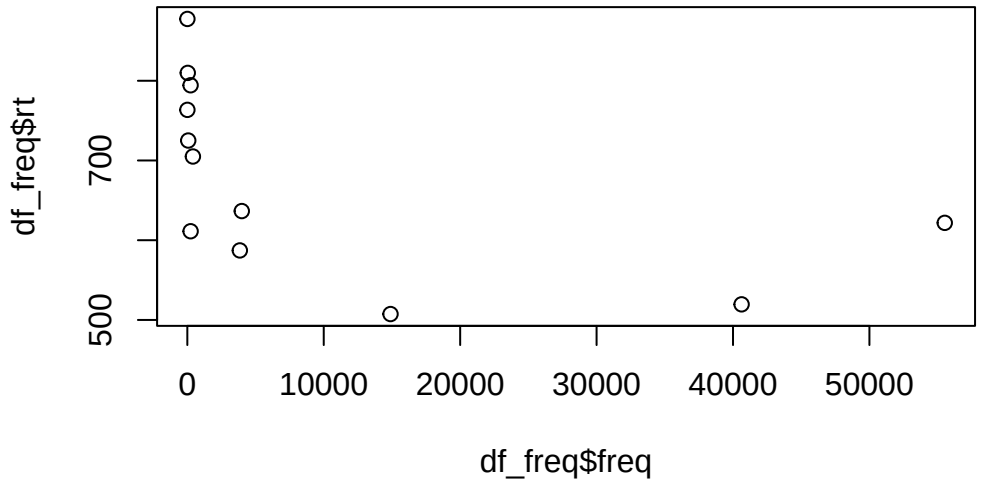
The `summary()` function provides summaries of each variable in a dataframe. For numeric variables, it will provide descriptive statistics for the centre and spread of the data (mean, median, quartiles). For categorical data, it will provide the count per category. For character variables, simply lists the number of observations.

```
summary(df_freq)
```

```
    word                freq                rt
Length:12          Min.   :    4.0   Min.   :507.4
Class :character   1st Qu.:   57.5   1st Qu.:605.2
Mode  :character   Median :  325.0   Median :670.8
                   Mean   : 9990.2   Mean   :679.9
                   3rd Qu.: 6717.8   3rd Qu.:771.2
                   Max.   :55522.0   Max.   :877.5
```

We see `freq` has a pretty big range, from 4 to 55522. `rt` has a range of 507.38 to 877.53, with an average reaction time of 679.9. Let's now get an overview of the relationship between `freq` and `rt`.

```
plot(df_freq$freq, df_freq$rt)
```



We see there are a lot of frequency values below roughly 400, and these seem to have higher reaction times than those with a higher frequency value. Let's fit these data to our first linear model to explore this effect of frequency on reaction times.

2.1.2 `lm()`

The the `lm()` function fits simple linear models. As arguments it takes a formula and a dataset, at minimum, as in equation 2.2.

$$lm(outcome \sim 1 + predictor,\ data\ =\ df\_name) \tag{2.2}$$

The `lm()` function formula syntax can be read as: `outcome` predicted by the intercept (`1` is a placeholder for the intercept) and predictor. The intercept is included by default, so if you omit the `1` the intercept is still included in the formula. If you wanted to remove the intercept (which you often don't), you could replace `1` with `0`.

2.1.2.1 Running a model

Before we add our predictor `freq`, let's see what our model looks like without it. We can write it as:

```
lm(rt ~ 1, data = df_freq)
```

But it's useful to save the model as an object so that we can call on it later. It's often a good idea to have informative prefixes to your objects

```
fit_rt_1 <- lm(rt ~ 1, data = df_freq)
```

> Object naming
>
> You may have wondered what the letters `df` are for when loading in our data set as `df_freq`. These letters stand for 'data frame', and serve as a reminder of what exactly that object in our environment is. We might also have wanted to plot the frequency data, in which case we could call save the plot as `fig_freq` or `plot_freq`. Here we are saving our model as `fit_rt_1`, using 'fit' to signal that this object is a model fit. You could also save it as `mod_freq_1`, `lm_freq_1`, or whatever you see fit. This simply helps keep our environment structured, which will become useful when you begin working with multiple datasets at a time.

2.1.2.2 Model ouput

Now that we've saved our model in our Enrivonement, we can call it by name. Printing just the model gives us the formula and the coefficients.

```
# print model
fit_rt_1
```

```
Call:
lm(formula = rt ~ 1, data = df_freq)

Coefficients:
(Intercept)
      679.9
```

Recall that the `intercept` and `slope` are called `coefficients`. Why do we only see `Intercept`? Because we didn't include any predictors in our model. This output isn't very dense, however. We typically use the `summary()` function to print full model outputs.

```
summary(fit_rt_1)
```

```
Call:
lm(formula = rt ~ 1, data = df_freq)

Residuals:
     Min       1Q   Median       3Q      Max
-172.537  -74.677   -9.137   91.296  197.613

Coefficients:
            Estimate Std. Error t value       Pr(>|t|)
(Intercept)   679.92      34.02   19.99 0.000000000538 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 117.8 on 11 degrees of freedom
```

We see a lot more information here.

> **`broom` package**
>
> The `broom` package has some useful functions for printing model outputs
>
> - `tidy()` produces a `tibble` (type of dataframe) of the `coefficients`
> - `glance()` produces goodness of fit measures (which we won't discuss)

The outputs from `tidy()` and `glance()` can be fed into `kable` and/or `kable_styling()` to create formatted tables

```
tidy(fit_rt_1)
```

```
# A tibble: 1 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)     680.      34.0      20.0 5.38e-10
```

```
glance(fit_rt_1)
```

```
# A tibble: 1 x 12
  r.squared adj.r.squared sigma statistic p.value    df logLik   AIC   BIC
      <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>
1         0             0  118.        NA      NA    NA  -73.7  151.  152.
# i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

`augment()` adds model values as columns to your dataframe (e.g., useful for plotting observed vs. fitted values).

```
augment(fit_rt_1, data = df_freq) %>% summary()
```

2.2 Interpreting model output

- let's take a closer look at our model summary

```
summary(fit_rt_1)
```

```
Call:
lm(formula = rt ~ 1, data = df_freq) #<1>

Residuals:
     Min       1Q   Median       3Q      Max
-172.537  -74.677   -9.137   91.296  197.613   #<2>

Coefficients:
            Estimate Std. Error t value       Pr(>|t|)    #<3>
```

```
(Intercept)   679.92      34.02   19.99 0.000000000538 *** #<4>
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 #<5>

Residual standard error: 117.8 on 11 degrees of freedom #<6>
```

① formula repetition
② residuals: differences between observed values and those predicted by the model
③ names for columns `Estimates`, `standard error`, `t-value`, `p-value` (`Pr(>|t|)`)
④ Intercept (b_0)
⑤ Significance codes
⑥ R^2, a measure of model fit (squared residuals); percentage of variance in the data shared with the predictor (higher numbers are better…this is pretty low)

2.2.0.1 Intercept

Our intercept is roughly 679.9 milliseconds; what does this number represent?

```
# print model intercept?
coef(fit_rt_1)['(Intercept)']
```

```
(Intercept)
   679.9167
```

```
# print data mean
mean(df_freq$rt)
```

```
[1] 679.9167
```

The intercept corresponds to the mean reaction time value. Let's explore this.

2.2.0.1.1 Intercept significance

In the model output, the intercept seems to be significant (indicated with a low p-value, and ***). What does this *mean*? Significance pretty much tells us if a number is equal to (or not statistically significantly different from) 0. So this tells us that the intercept (i.e., the mean reaction time) is different from 0. How do we interpret this? In most cases we don't. Whether or not the intercept is significantly different from 0 this isn't interesting or even theoretically relevant, as reaction times *shouldn't* be near 0, so neither should their mean. This is also true for formant frequencies, reading times, and other types of continuous linguistic data.

2.2.0.2 Standard Error

Standard error takes both the variability in our data and the sample size into account. The equation for standard error is:

$$SE = \frac{\hat{\sigma}}{\sqrt{n}} \tag{2.3}$$

where σ is the standard deviation, and n is the sample size. As a refresher, the equation for standard deviation (2.4) is the square root of the sum of all squared deviances from the mean ($\sum_{i=1}^{n}(x_i - \hat{\mu})^2$) divided by the sample size -1. Don't stress about the math for now, but it's helpful to try to understand where there values come from and what they represent.

$$\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{\mu})^2}{n-1}} \tag{2.4}$$

2.2.0.3 t-values

Simple linear regression is equivalent to a t-test. The one-sample t-test corresponds to an intercept-only.

```
df_freq %>%
t.test(rt ~ 1, data = .)
```

```
    One Sample t-test

data:  rt
t = 19.988, df = 11, p-value = 0.000000000538
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 605.0461 754.7872
sample estimates:
mean of x
 679.9167
```

```
df_freq %>%
lm(rt ~ 1, data = .) %>%
  tidy() %>%
  mutate_if(is.numeric, round, 10)
```

```
# A tibble: 1 x 5
  term        estimate std.error statistic      p.value
  <chr>          <dbl>     <dbl>     <dbl>        <dbl>
1 (Intercept)     680.      34.0      20.0 0.0000000005
```

The *real* power of linear regression is coming tomorrow and in January…multiple regression and mixed models. But for now, it's important to remember that the larger the t-value, the smaller the p-value. But more important is to not rely too heavily on p-values, as such black-and-white classifications have proven a poor substitute for understanding our data and our models.

2.2.0.4 p-values

> A word on t-values and p-values
>
> *t*-values quantify the *difference* between population means.
> *p*-values quantify the probability of obtaining a result equal to or greater than what was observed, given the assumption of no effect (the null hypothesis).
> If the null hypothesis were true, we would expect no effect (a flat line). If we have a lot of evidence/are confidence that there is an effect (the line (slope) is in fact *not* flat), then it would be unlikely that we would find such a result under the assumption that there is no effect (the line actually *is* flat) i.e., the null hypothesis. This is reflected in a small p-value.

2.2.0.5 Plotting `rt ~ 1`

- Figure 2.1 shows the intercept (red dot) amongst the observed data (black dots)
  - along the x-axis we have abstract numerical units (the values don't mean anything)
  - what would the values of the intercept be?

2.3 Adding a fixed effect (slope)

Now let's include a predictor, which will give us a *slope*. The slope represents the change in y (DV: `rt`) when we move 1-unit along y (IV: `freq`). In other words, it tells us the *effect* our IV has on the DV. Let's first plot the data:

```
df_freq |>
  ggplot() +
```

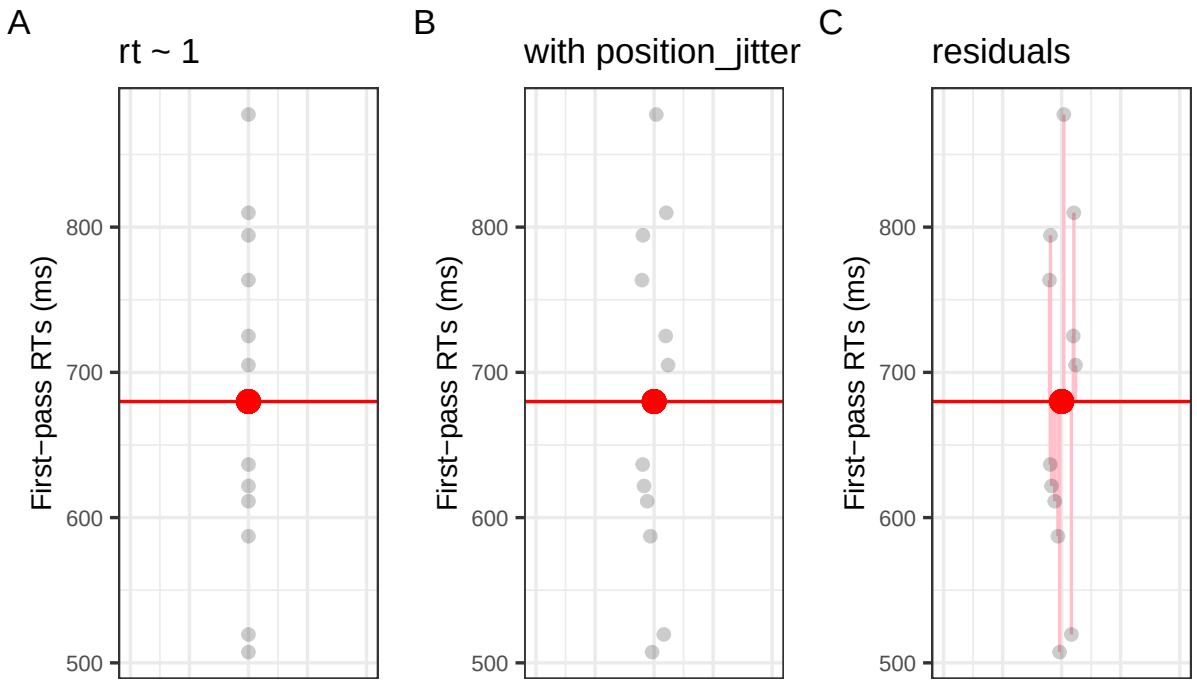
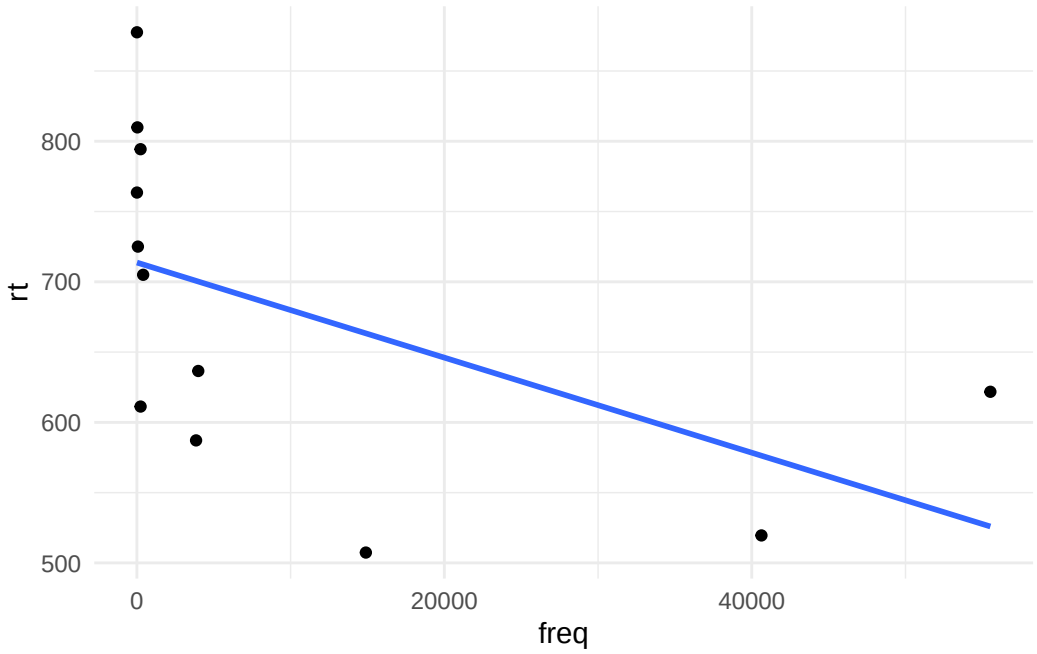


Figure 2.1: Visualisation of 'rt ~ 1': observed values (black) and mean (intercept; red). Residuals would be the distance from each black dot to the y-value of the read dot

```
aes(x = freq, y = rt) +
geom_point() +
geom_smooth(method = "lm", se = FALSE)
```



2.3.1 Fit model (treatment contrasts)

```
# fit simple linear model
fit_rt_freq <- lm(rt ~ freq, data = df_freq)
```

2.3.1.1 Model summary

```
summary(fit_rt_freq)
```

```
Call:
lm(formula = rt ~ freq, data = df_freq)

Residuals:
     Min       1Q   Median       3Q      Max
-155.947  -73.141    2.117   85.050  163.837

Coefficients:
             Estimate Std. Error t value     Pr(>|t|)
(Intercept) 713.706298  34.639105   20.60 0.0000000016 ***
```

```
freq         -0.003382   0.001699   -1.99       0.0746 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 104.6 on 10 degrees of freedom
Multiple R-squared:  0.2838,    Adjusted R-squared:  0.2121
F-statistic: 3.962 on 1 and 10 DF,  p-value: 0.07457
```

2.3.1.2 Intercept

The intercept in our last model was the mean reaction time. But now it's a different value.

```
# print model intercept
coef(fit_rt_freq)['(Intercept)']
```

```
(Intercept)
   713.7063
```

```
# print data mean
mean(df_freq$rt)
```

```
[1] 679.9167
```

Our intercept is no longer the grand mean of first-pass reading times…what is it?

2.3.1.3 Slope

Our slope was our slope -0.0033823. What does this correspond to?

```
# print slope
coef(fit_rt_freq)['freq']
```

```
       freq
-0.003382289
```

This is the change in y (our DV `rt`) for a 1-unit change in x (our IV: `freq`). So when we move up 1 unit in frequency, reaction times decrease by -0.0033823. Whether or not it makes sense to consider this number depends on the measurement unit your data is in, e.g., a unit change from one millimeter or one meter will have a drastically different slope value (say, for age), but the actual slope will be the exact same.

```
heights_m <- c(1.71, 1.56, .9, 2.06, 1.63)
heights_cm <- c(171, 156, 90, 206, 163)
heights_mm <- c(1710, 1560, 900, 2060, 1630)
year <- c(22,15,10,26,18)
months <- c(22,15,10,26,18)*12
days <- c(22,15,10,26,18)*365

df_heights_age <- cbind(year, months, days, heights_mm, heights_cm, heights_m) |> as.data.
  pivot_longer(
    cols = c(heights_mm, heights_cm, heights_m),
    names_to = "unit",
    values_to = "height"
  ) |>
  pivot_longer(
    cols = c(year, months, days),
    names_to = "unit_age",
    values_to = "age"
  )




lm(heights_mm ~ year)
```

```
Call:
lm(formula = heights_mm ~ year)

Coefficients:
(Intercept)         year
     396.62        64.58
```

```
lm(heights_cm ~ days)
```

```
Call:
lm(formula = heights_cm ~ days)

Coefficients:
(Intercept)         days
   39.66230      0.01769
```

```
  lm(heights_m ~ months)
```

```
Call:
lm(formula = heights_m ~ months)

Coefficients:
(Intercept)       months
   0.396623     0.005382
```

```
  lm(heights_mm ~ year)
```

```
Call:
lm(formula = heights_mm ~ year)

Coefficients:
(Intercept)         year
     396.62        64.58
```

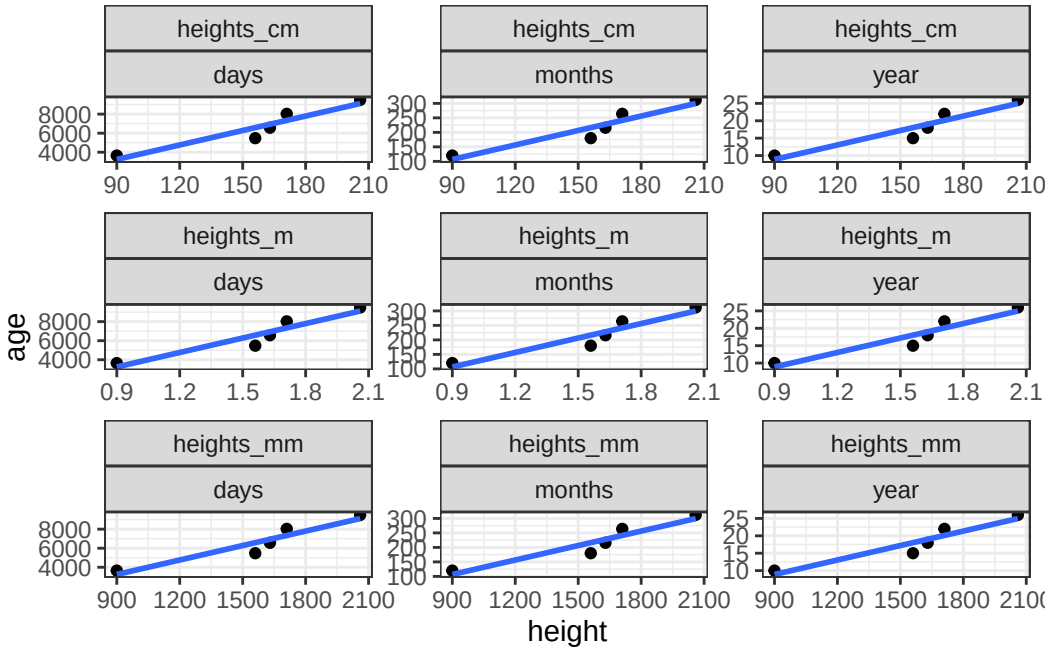
```
  lm(heights_cm ~ year)
```

```
Call:
lm(formula = heights_cm ~ year)

Coefficients:
(Intercept)         year
     39.662        6.458
```

```
  lm(heights_m ~ year)
```

```
Call:
lm(formula = heights_m ~ year)

Coefficients:
(Intercept)         year
    0.39662      0.06458
```

```r
ggplot(data = df_heights_age) +
  aes(x = height, y = age) +
  facet_wrap(unit ~ unit_age, scales = "free") +
  geom_point() +
  geom_smooth(method = "lm", se = F) +
  theme_bw()
```



2.4 Model assumptions

Now that we've fit a model and understand the output, it's time to think about whether this model is a good fit for our data. We first have to understand some assumptions that need to met in regression modelling. Importantly, these assumptions reate to the *residuals* of our model, not the raw data points themselves. The two assumptions we'll focus on for now are

the assumptions of *normality* of the residuals, and the constant *variance* of the residuals. Both assumptions are often diagnosed visually, so it takes some practice to learn what looks right.

2.4.1 Normality

When a model satisfies the normalit assumption, its *residuals* (i.e., the difference between the *fitted* and *observed* values) will be approximately normally distributed. Normality is typically visualised using a histogram (Figure 2.2 A) and/or a quantile-quantile (Q-Q) plot (Figure 2.2 B).

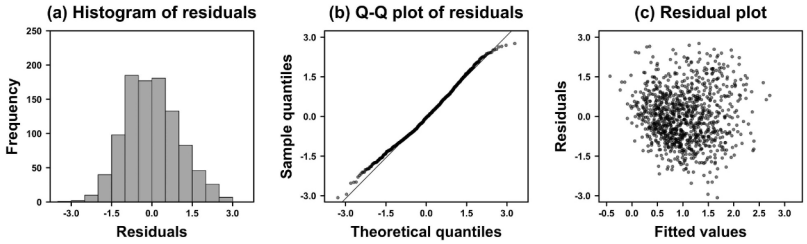


Figure 6.1. (a) Histogram, (b) Q-Q plot, and (c) residual plot of `icon_mdl`

Figure 2.2: Image source: Winter (2019) (all rights reserved)

> **Note**
>
> Winter (2019)'s description of how QQ plots are generated (p. 110):
> *To create this plot, every residual is transformed into a percentile (or quantile) […] The question the Q-Q plot answers is: what is the corresponding numerical value of the 13.8th percentile on the normal distribution? If the values are the same, they will fit on a straight line, which indicates that the two distributions (the distribution of the residuals and the theoretical normal distribution) are very similar.*

2.4.2 Constant variance

When a model satisfies the constant variance assumption (also called *homoscedasticity*, or the absence of *heteroscedasticity*), the spread of residuals will be equal across the regression line. This is typically visualised using a residual plot, which should look like a blob (Figure 2.2 C).

2.4.3 Visualising model assumptions

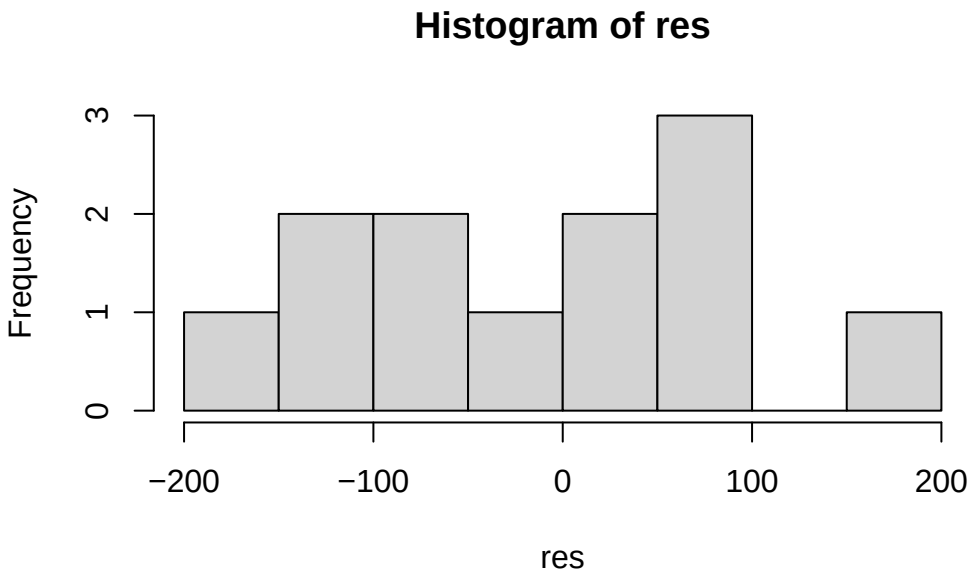
Let's plot our residuals to assess whether our model satisfies the assumptions of normality and constant variance.

2.4.3.1 Histogram

We can do this how it's done in Winter (2019) (in Ch. 6, p. 110-111), by first extracting the residuals from the model and then fitting them them using the base R function `hist()`.

```
# extract residuals
res <- residuals(fit_rt_freq)
```

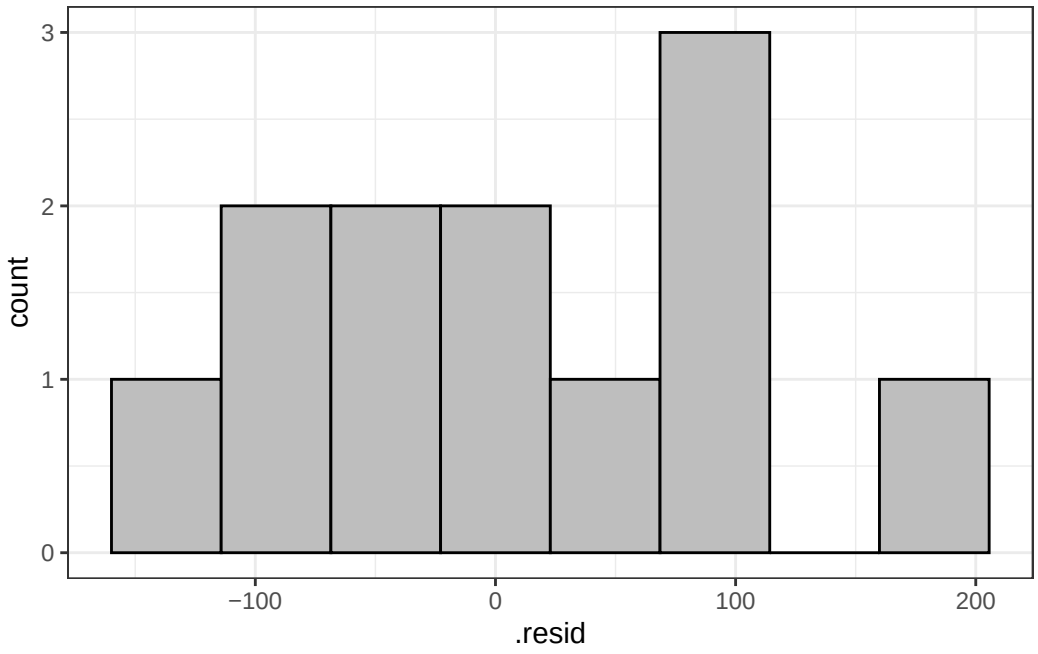
```
# plot histogram
hist(res)
```



Or, we can use the `augment()` function from `broom` to append model values to our original data frame, and then feed this into `ggplot()` from `ggplot2` (or even feed it into `hist()`).

```
# or, add to df
df_freq <- broom::augment(fit_rt_freq, df_freq)
```

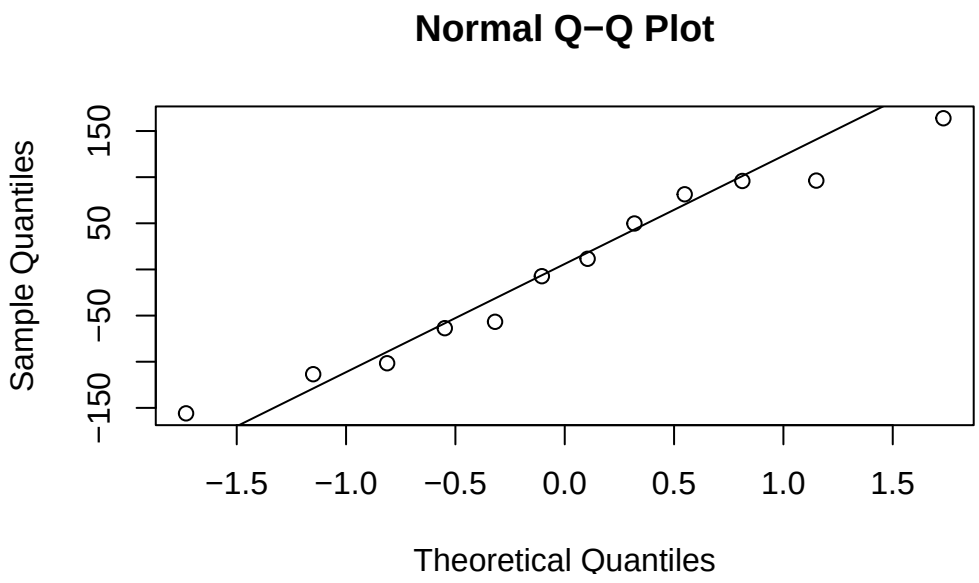
```r
# and create ggplot
df_freq |>
  ggplot() +
  aes(x = .resid) +
  geom_histogram(bins = 8, fill = "grey", colour = "black") +
  theme_bw()
```



2.4.3.2 Q-Q plot

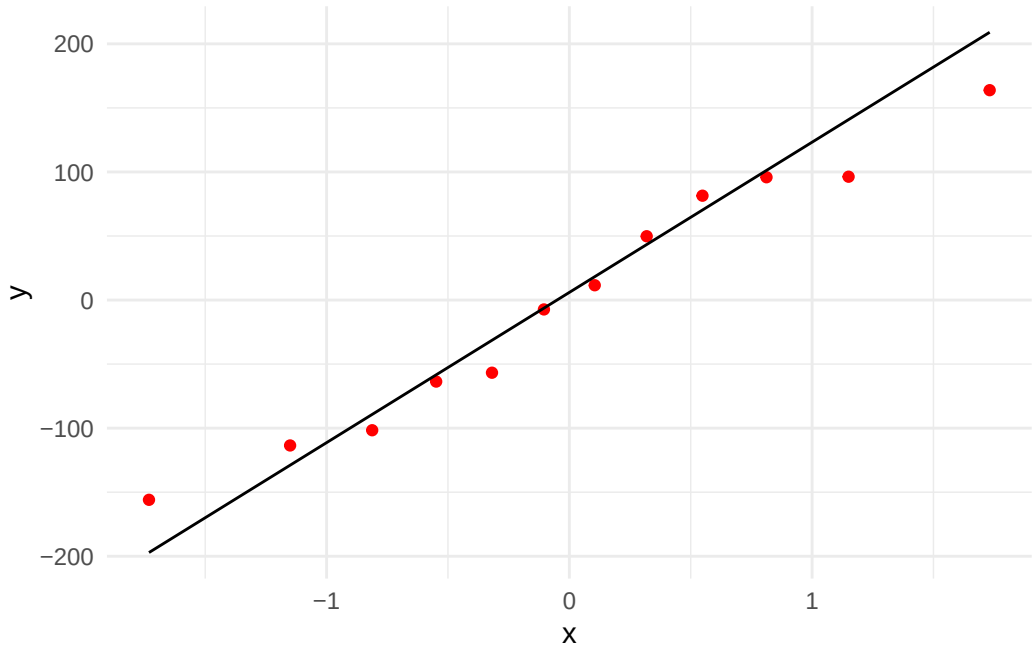
Again, we can do it Bodo's way:

```r
qqnorm(res)
qqline(res)
```



Or using `augment()` and `ggplot()`.

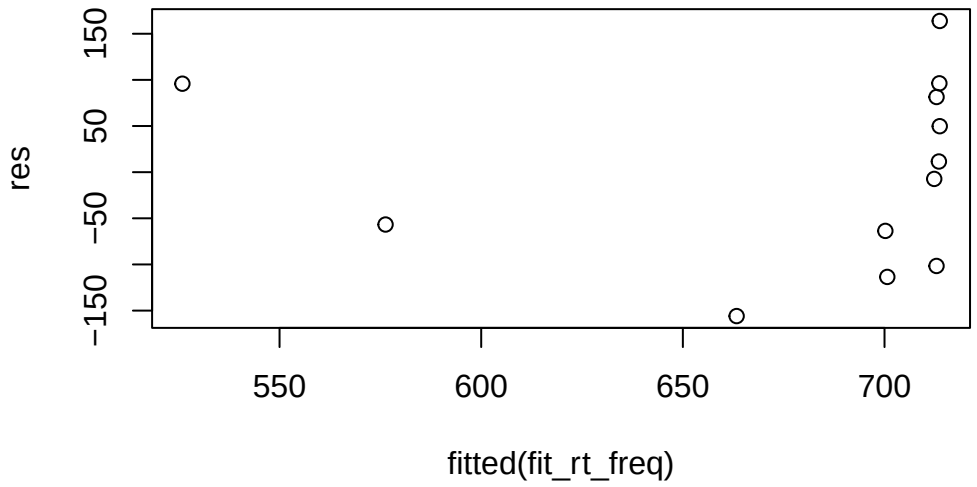
```
df_freq |>
  ggplot() +
  aes(sample = .resid) +
  geom_qq(colour = "red") +
  geom_qq_line()
```



2.4.3.3 Residual plot

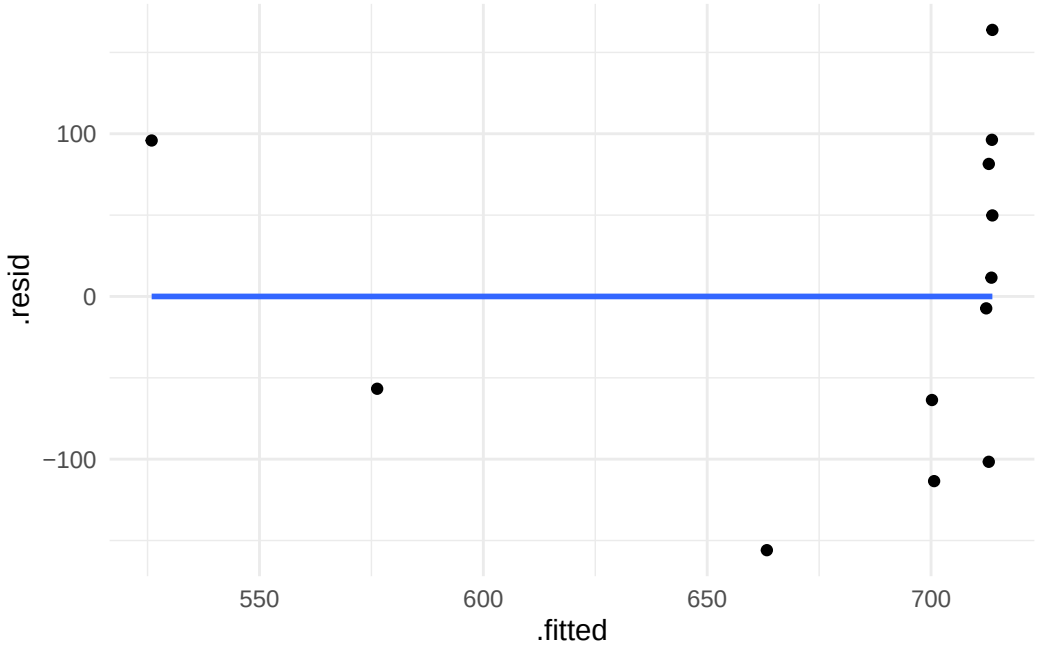
Bodo's way:

```
plot(fitted(fit_rt_freq), res)
```



Or with `ggplot`:

```
df_freq |>
  ggplot() +
  aes(x = .fitted, y = .resid) +
  geom_point() +
  geom_smooth(method = "lm", se = F)
```



2.4.4 `performance` package

I like to use the `performance` package to visualise model fit (Lüdecke et al., 2021).

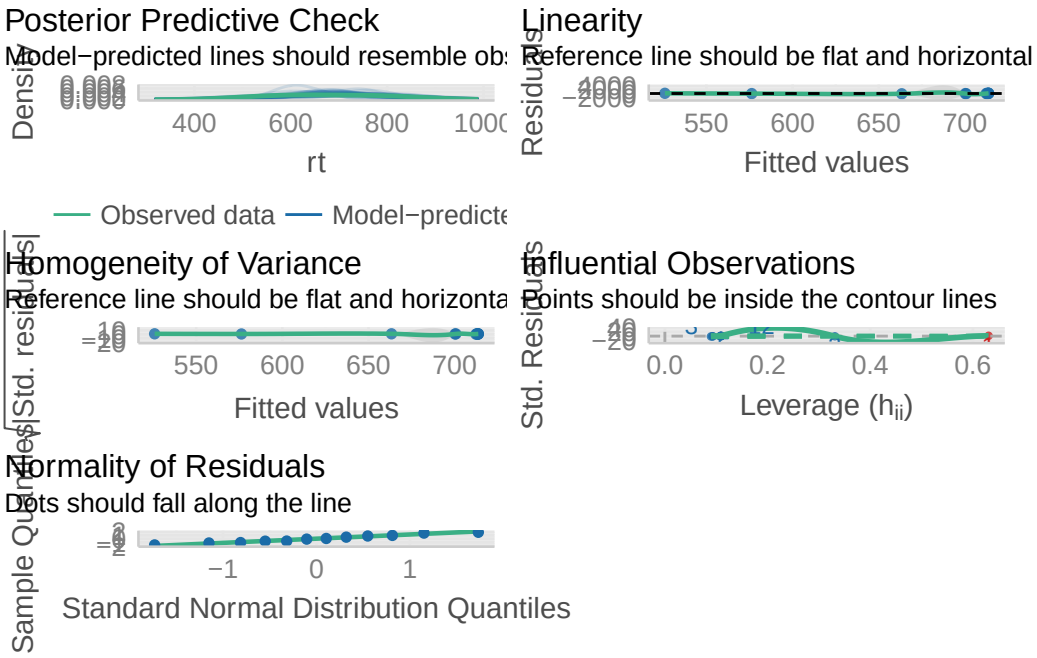
```
performance::check_normality(fit_rt_freq)
```

```
OK: residuals appear as normally distributed (p = 0.702).
```

```
performance::check_heteroscedasticity(fit_rt_freq)
```

```
OK: Error variance appears to be homoscedastic (p = 0.980).
```

```
performance::check_model(fit_rt_freq)
```



2.4.4.1 Coefficients table with `summary()`

```
> summary(fit_rt_freq)

Call:
lm(formula = rt ~ lifetime, data = df_freq, subset = rt > 0) #<1>

Residuals:                                                   #<2>
    Min      1Q  Median      3Q     Max
-228.99 -109.29  -26.99   58.86  777.71

Coefficients:
            Estimate Std. Error t value Pr(>|t|)                #<3>
(Intercept)  309.142      6.259  49.394 <0.0000000000000002 ***          #<4>
lifetime1     31.701     12.517   2.533               0.0116 *              #<5>
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 57.46 on 541 degrees of freedom
Multiple R-squared:  0.01172,   Adjusted R-squared:  0.00989    #<6>
F-statistic: 6.414 on 1 and 541 DF,  p-value: 0.0116            #<7>
```

① formula
② Residuals: differences between observed values and those predicted by the model
③ Names for columns Estimates, SE, t-value, p-value
④ Intercept (b_0), i.e., value of y (first-pass) with a move of one unit of x (lifetime)
⑤ Slope (b_1), i.e., change in first fixation going from `dead` to `living`
⑥ Output from an ANOVA

- what is the **intercept**?
- is the **slope** positive or negative?
  - what is it's value?
- this is what the slope would look like:

Exploring the model

```
# how many observed values did we enter into the model?
df_freq |>
  nrow()
```

```
[1] 12
```

```
# how many observed values did we enter into the model?
length(fitted(fit_rt_freq))
```

```
[1] 12
```

Exploring the model: residuals

```
# what do our FITTED values look like?
head(fitted(fit_rt_freq))
```

```
       1        2        3        4        5        6
525.9148 576.2873 663.3271 700.2042 700.6845 712.3229
```

```
# what do our OBSERVED values look like?
head(df_freq$rt)
```

```
[1] 621.77 519.56 507.38 636.56 587.18 705.00
```

```
# what is the difference between the FITTED and OBSERVED values?
head(df_freq$rt) - head(fitted(fit_rt_freq))
```

```
        1           2           3           4           5           6
 95.855154  -56.727276 -155.947103  -63.644200 -113.504485   -7.322942
```

```
# what are our RESIDUALS?
head(residuals(fit_rt_freq))
```

```
        1           2           3           4           5           6
 95.855154  -56.727276 -155.947103  -63.644200 -113.504485   -7.322942
```

Exploring the model

- what were our coefficients?

```
coef(fit_rt_freq)
```

```
  (Intercept)          freq
713.706297951  -0.003382289
```

- what would be our predicted reaction time for a word with frequency of 0?

```
coef(fit_rt_freq)['(Intercept)'] + coef(fit_rt_freq)['freq'] * 0
```

```
(Intercept)
   713.7063
```

- ignore the `(Intercept)` label here, `R` just takes the first label when performing an operation on 2 vectors
- what is the mean of our predictor coded as `+0.5`?

```
coef(fit_rt_freq)['(Intercept)'] + coef(fit_rt_freq)['freq'] * 5000
```

```
(Intercept)
   696.7949
```

Table A1. Correspondences between some significance tests and linear models

Significance test	*Linear model*	*Description*
`t.test(y ~ pred, paired = FALSE)`	`lm(y ~ pred)`	An unpaired *t*-test corresponds to a linear model with a binary categorical predictor
`t.test(y, mu = 0)`	`lm(y ~ 1)`	One-sample *t*-test corresponds to an intercept-only model
`t.test(y ~ pred, paired = TRUE)`	`lm(diffs ~ 1)`	A paired *t*-test corresponds to an intercept-only model fitted on differences
`chisq.test(xtab)`	`glm(y ~ x, family = binomial)`	A chi-square test can be emulated with a logistic regression model

Table A2. Correspondences between ANOVAs and linear models

ANOVA	*Linear model*	*Description*
`aov(y ~ c3)`	`lm(y ~ c3)`	One-way ANOVA with three-level factor
`aov(y ~ c2 * c2)`	`lm(y ~ c2 * c2)`	2 x 2 ANOVA (two-way ANOVA)
`aov(y ~ c2 * c3)`	`lm(y ~ c2 * c3)`	2 x 3 ANOVA (and so on)
`aov(y ~ c2 * covariate)`	`lm(y ~ c2 * covariate)`	ANCOVA (analysis of covariance) with covariate (continuous predictor) and many other types of similar models

Instead of repeated measures ANOVA, you can use mixed models. The linear model framework allows much more complex random effects structures, thus giving the user more flexibility in expressing their theories.

Figure 2.3: Image source: Winter (2019) (all rights reserved)

2.5 Reporting your model

Section

2.6 Summary

- we saw that the equation for a straight line boils down to its intercept and slope
- we fit our first linear model with a categorical predictor

Important terms

Term	Definition
Coefficients	the slope and the intercept are coefficiens
Simple linear regression	linear regression with a single predictor and a continuous outcome variable
fitted values	predicted values
continuous variable	a variable that can have an infinite number of values (an example would be reading
dependent variable	what we measure; a.k.a. measure/outcome/response variable
independent variable	our predictor; a.k.a., predictor variable, fixed effects
coefficients	values of the intercept and slope of a lm() model
equation of a line	value of y = intercept + (slope*value of x)

Learning Objectives

Today we learned…

2.7 Task

Now it's your turn. Try to run the following `lm()` models:

1. total reading time at the *verb* region
2. total reading time at the *verb+1* region.

3 Continuous predictors

Regression for Linguists

This lecture is based on Ch. 5 (Correlation, Linear, and Nonlinear transformations) from Winter (2019).

Learning Objectives

Today we will learn...

- why and how to centre continuous predictors
- when and how to standardize continuous predictors
- why and how to log-transform continuous variables

Set-up environment

```
# suppress scientific notation
options(scipen=999)
```

We'll also need to load in our required packages. Hopefully you've already install the required packages (if not, go to **?@sec-software**).

```
# load libraries
pacman::p_load(
               tidyverse,
               here,
               broom,
               lme4,
               janitor,
               languageR)
```

Load data

```
df_freq <- read_csv(here("data", "ELP_frequency.csv")) |>
  clean_names()
```

Reminder of our variables:

```
summary(df_freq)
```

```
    word                freq                rt        
Length:12          Min.   :    4.0   Min.   :507.4  
Class :character   1st Qu.:   57.5   1st Qu.:605.2  
Mode  :character   Median :  325.0   Median :670.8  
                   Mean   : 9990.2   Mean   :679.9  
                   3rd Qu.: 6717.8   3rd Qu.:771.2  
                   Max.   :55522.0   Max.   :877.5  
```

Summary

In the last lectures we saw that the equation for a straight line boils down to its intercept and slope, and that linear regression fits a line to our data. This line results in predicted/fitted values, which fall along the line, and residuals, which are the difference between our observed values and the fitted values.

We also learned about two model assumptions: normality of residuals, and constant variance of residuals. We learned that we can plot these with histograms or Q-Q plots (normality), and residual plots (constant variance).

Now that we understand what a simple linear does, we can take a step back and focus on what we put into the model. So far we've looked at reaction times (milliseconds) as a function of word frequency. However, we don't typically feed raw continuous data into a model, because most continuous linguistic variables are not normally distributed, and so a straight line will not fit it very well (because there will be some large variance for higher values).

3.1 Linear transformations

Linear transformations refer to constant changes across values that do not alter the relationship between these values. For example, adding, subtracting, or multiplying by a constant value will not alter the difference between values. Think of the example in the last lecture on

the relationship between heights and ages as a function of the measurement unit: the relationship between all the values did not alter, because the difference between heights millimeters, centimeters, and meters is constant, as is the difference between ages in days, months, or years. We'll now look at some common ways of linearly transforming our data, and the reasons behind doing so.

3.1.1 Centering

Centering is typically applied to predictor variables. Centering refers to subtracting the mean of a variable from each value, resulting in each centered value representing the original value's deviance from the mean (i.e., a mean-deviation score). What would a centered value of 0 represent in terms of the original values?

Let's try centering our frequency values. To create a new variable (or alter an existing variable), we can use the `mutate()` function from `dplyr`.

```
# add centered variable
df_freq <-
  df_freq |>
  mutate(freq_c = freq-mean(freq))
```

This can also be done with base R, but it's a lot more verbose.

```
# add centered variable with base R
df_freq$freq_c <- df_freq$freq-mean(df_freq$freq)
```

Now let's fit our models.

```
# run our model with the original predictor
fit_rt_freq <-
  lm(rt ~ freq, data = df_freq)
```

```
# run our model with the centered predictor
fit_rt_freq_c <-
  lm(rt ~ freq_c, data = df_freq)
```

If we compare the coefficients from `fit_rt_freq` and `fit_rt_freq_c`, what do we see? The only difference is the intercept values: 713.706298 (uncentered) and 679.9166667 (centered).

```
mean(df_freq$rt)
```

```
[1] 679.9167
```

The intercept for a centered continuous predictor variable corresponds to the mean of a continuous response variable. This is crucial in interpreting interaction effects, which we will discuss tomorrow. For more detail on interpreting interactions, see Chapter 8 in Winter (2019) (we won't be discussing this chapter as a whole).

> Centering interval data
>
> If you have interval data with a specific upper and lower bound, you could alternatively subtract the median value. In linguistic research, this is most typically rating scale data. For example, if you have a dataset consisting of ratings from 1-7, you can centre these ratings by subtracting 4 from all responses. A centred response of -3 would correspond to the lowest rating (1), and of +3 to the highest rating (7), which 0 would correspond to a medial rating (4). This can also be helpful in plotting, as there is no question as to whether 1 or 7 was high or low, because all ratings are now centred around 0 (and negative numbers correspond to our intuition of low-ratings).

3.1.2 Standardizing (*z*-scoring)

We can also standardize continuous predictors by dividing centered values by the standard deviation of the sample. Let's look at our frequency/reaction time data again.

First, what are our mean and standard deviation? This will help us understand the changes to our variables as we center and stardardize them.

```
mean(df_freq$freq)
```

```
[1] 9990.167
```

```
sd(df_freq$freq)
```

```
[1] 18558.69
```

What are the first six values of `freq` in the original scale?

```
df_freq$freq[1:6]
```

```
[1] 55522 40629 14895  3992  3850   409
```

What are the first six values of `freq_c` in the centered scale? These should be the values of `freq` minus the mean of `freq` (which we saw above is 9990.1666667).

```
df_freq$freq_c[1:6]
```

```
[1] 45531.833 30638.833  4904.833 -5998.167 -6140.167 -9581.167
```

Now, let's create our standardised z-scores for frequency by dividing these centered values by the standard deviation of `freq` (which will be the same as the standard deviation of `freq_c`), and which we saw is 18558.6881679. Again, this can be done with `mutate()` from `dplyr`, or by using base R syntax.

```
# standardise using the tidyverse
df_freq <-
  df_freq |>
  mutate(freq_z = freq_c/sd(freq))
```

```
# standardize with base R
df_freq$freq_z <- df_freq$freq_c/sd(df_freq$freq)
```

```
head(df_freq)
```

```
# A tibble: 6 x 5
  word      freq    rt freq_c freq_z
  <chr>    <dbl> <dbl>  <dbl>  <dbl>
1 thing    55522  622. 45532.  2.45
2 life     40629  520. 30639.  1.65
3 door     14895  507.  4905.  0.264
4 angel     3992  637. -5998. -0.323
5 beer      3850  587. -6140. -0.331
6 disgrace   409  705  -9581. -0.516
```

Correlation

3.2 Non-linear transformations

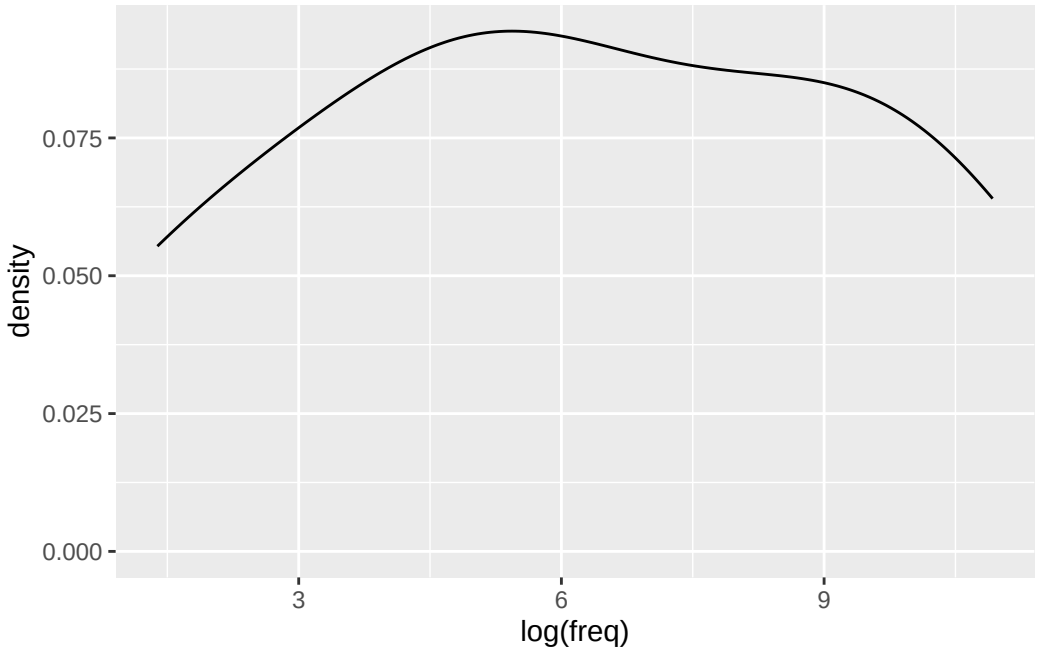
This is really the meat and potates of dealing with continuous variables (depending on your subfield). In linguistic research, and especially experimental research, we often deal with

continuous variables truncated/bound at 0. Reaction times, reading times and formant frequencies are all examples of such types of data: there is no such thing as a negative reading time or fundamental frequency. The problem with these types of data is that they are almost never normally distributed, which has implications for the normality of residuals for any line that tries to fit to these data. Very often, this type of data will have a 'positive skew', or a long tail off to the right (assuming larger values are plotting to the right). This shape is not symmetrical, meaning that the residuals tend to be much larger for larger values. It is also often the case that these very large, exceptional values will have a stronger influence on the line of best fit, leading to the coefficient estimates that are "suboptimal for the majority of data points" [@Baayen (2008); p. 92]. How do we deal with this nonnormality? We use non-linear transformations, the most common of which is the log-transformation.

3.2.1 Log-transformation

Let's look at our reaction time data again. We'll log-transform our reaction time data and frequency data. Note that in Winter (2019), frequency is transformed using log to the base 10 for interpretability, but we'll stick to the natural logarithm.

```
df_freq |>
  ggplot() +
  aes(x = log(freq)) +
  geom_density()
```



```r
df_freq <-
  df_freq |>
    mutate(rt_log = log(rt),
           freq_log = log(freq))
```

```r
lm(rt_log ~ freq_log, data = df_freq) |> tidy()
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)   6.79     0.0611     111.   8.56e-17
2 freq_log     -0.0453   0.00871     -5.20 4.03e- 4
```

```r
# or, log-transform directly in the model syntax
lm(log(rt) ~ log(freq), data = df_freq) |> tidy()
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)   6.79     0.0611     111.   8.56e-17
2 log(freq)    -0.0453   0.00871     -5.20 4.03e- 4
```

Learning Objectives

Today we learned...

- why and how to centre continuous predictors
- when and how to standardize continuous predictors
- why and how to log-transform continuous variables

Important terms

Term	Definition	E
Centering	type of linear transformation	‘
standardizing	linear transformation (applied for multiple continuous predictors)	d
log-transformation	non-linear transformation for positively skewed continuous variables	l

Take-home messages

Continuous data are often transformed before fitting a model to this data. Linear transformations, like adding or multiplying all values by a single value, are often performed on continuous predictors by means of centring and standardizing (when there are multiple continuous predictors). Non-linear transformations are often performed on continuous data with a positive skew (a few values much larger than the majority) in order to satisfy the normality assumption. Although the normality assumption refers to the normality of *residuals*, the distribution of the data will have implications for the distribution of the residuals. The most common non-linear transformation is the log-transformation (the inverse of the exponential), which shrinks values, especially making big numbers smaller. This has the result of squeezing big numbers towards smaller numbers, reducing the spread in the distribution (e.g., the log of 3 is 1.0986123, the log of 30 is 3.4011974, and the log of 30 is 5.7037825).

What to do with this information? If you have continuous data truncated at 0 (with no upperbound, e.g., reaction time data or fundamental frequency), visualise the data (histogram and Q-Q plot) in order to check its distribution. If it is not normally distributed, you will likely want to log-transform it. Is this data your *response* variable? Then that is all you will likely want to do. Is this data a *predictor* variable? Then you will want to centre it (subtract the mean of this variable from all values). Do you have more than one continuous predictor variable? Then standardizing these variables will facilitate the interpretation of interaction effects (we'll talk about these soon).

3.3 Task

3.4 Assessing assumptions

1. Re-run the models `fit_rt_freq`, `fit_rt_freq_c`, and `fit_log`
2. Produce diagnostic plots for each of them (histograms, Q-Q plots, residual plots)
3. Interpret the plots

3.5 Model comparison

1. Use the `glance()` function to inspect the R^2, AIC, and BIC of each model.
2. Which is the best fit? Why?

Part III

Day 2: Multiple regression

4 Multiple Regression

Regression for Linguists

Summary

- we saw that the equation for a straight line boils down to its intercept and slope
- we fit our first linear model with a continuous predictor

Learning Objectives

Today we will learn...

- what multiple regression is
- how to include multiple predictor variables
- how to interpret slopes in multiple regression
- how to interpret interaction effects
- about the assumption of the absence of collinearity

Set-up environment

```
# suppress scientific notation
options(scipen=999)
```

We'll also need to load in our required packages. Hopefully you've already install the required packages (if not, go to **?@sec-software**).

```
# load libraries
pacman::p_load(
               tidyverse,
```

```
               here,
               broom,
               janitor,
               languageR)
```

Load data

We'll use the full dataset of the frequency data.

```
df_freq_full <-
  read_csv(here("data", "ELP_full_length_frequency.csv")) |>
  clean_names() |>
  mutate(freq = 10^(log10freq), # inverse log10
         freq_log = log(freq)) |>  # use natural logarithm
  relocate(word, rt, length, freq, freq_log)
```

We have 4 variables:

- `word`
- `length`
- `rt`
- `freq`
- `freq_log`
- `log10freq`

4.1 Multiple regression

So far we've worked with simple linear models, which fit a model to a predictor and response variable. These models do not differ so greatly from a one- or two-sample t-test (for a categorical predictor) or Pearson's r (for a standardised continuous predictor). You might be wondering then we would bother with linear regression. One reason is that it allows us to include *multiple* predictors in our models, which still boils down to modeling the mean, but while condintioning the mean on multiple variables at once.

Recall the equation of a line (4.1), which states that any value of y equals the intercept (b_0) plus the corresponding value of x multiplied by the slope ($b_1 x$), plus the error, which are our residuals (e). In multiple regression, we can include more than one slope (4.2).

$$y = b_0 + b_1 x + e \tag{4.1}$$
$$y = b_0 + b_1 x + b_2 x + ... + e \tag{4.2}$$

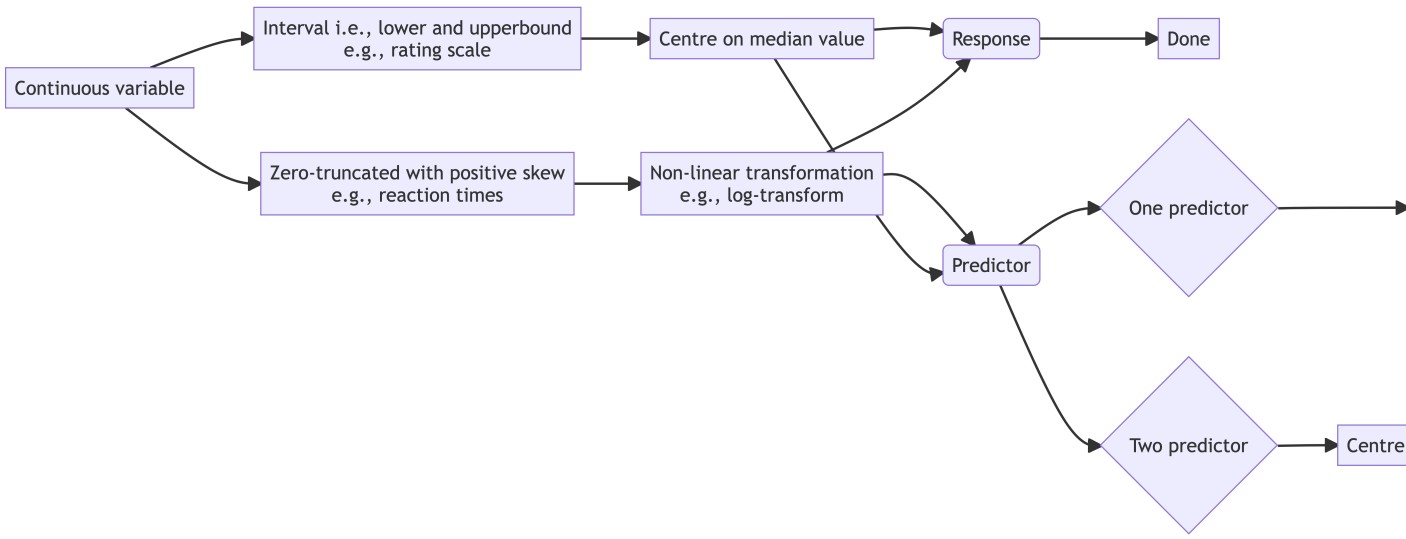


Figure 4.1: Flowchart of common steps for linear and non-linear transformations of continuous variables. Such decision trees are not a one-size-fits-all solution and cannot replace critical thinking and understanding of your data.

4.1.1 One predictor

Let's re-run our simple model with this dataset. Let's keep reaction times in the raw milliseconds for now for interpretability.

```r
fit_freq_full <-
  lm(rt ~ log(freq), data = df_freq_full)
```

```r
tidy(fit_freq_full)
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic p.value
  <chr>          <dbl>     <dbl>     <dbl>   <dbl>
1 (Intercept)    907.      1.09       828.       0
2 log(freq)      -37.5     0.262     -143.       0
```

We see there is a decrease in reaction times (-37.5 milliseconds) for a 1-unit increase in log frequency. Let's look at the model fit using `glance()`.

```r
glance(fit_freq_full)$r.squared
```

```
[1] 0.3834186
```

We see that the R-squared is 0.383, meaning our model describes 38% of the variance in response times. We can't be sure that this described variance is due solely to frequency, however. Our models only know what we tell them! Other effects that are correlated with frequency might be conflating the frequency effect, e.g., more frequent words tend to be shorter (**zipf_1949?**). Let's expand our model to include word length [4.3].

$$y = b_0 + b_1 * logfrequency + b_2 * wordlength \tag{4.3}$$

4.1.2 Adding a predictor

Let's add `length` as a predictor to our model.

```r
fit_freq_mult <-
  lm(rt ~ log(freq) + length, data = df_freq_full)
```

```r
tidy(fit_freq_mult) |> select(term, estimate)
```

```
# A tibble: 3 x 2
  term        estimate
  <chr>          <dbl>
1 (Intercept)    748.
2 log(freq)      -29.5
3 length          19.5
```

We see that length is also a significant predictor of reaction times, with an increase in word length (+1 letter) corresponds to a 20ms increase in reaction times. Our intercept is also now 748ms, instead of 907ms. The 907ms intercept corresponds to the prediction for reaction times to a word with 0 log frequency and 0 word length, but this is not very interpretable. If we were to center both prdictors, the intercept would be the reaction time for a wrd with average frequency and average length.

The slope for log frequency has also changed: from -37.5 to -29.5. This change tells us that some of the effect in our first model was confounded with length, as controlling for length weakens the effect of frequency.

```
glance(fit_freq_mult)$r.squared
```

```
[1] 0.4872977
```

We also see that including `length` increases the variance described by our model, reflected in the R-squared values (0.4872977 instead of 0.3834186.

4.2 Standardising our predictors

Recall that, when we have multiple continuous predictors, standardising them can help their interpretation, as their slopes are comparable. We could achieve this by centering each variable and then dividing by the standard deviation, or we could use the `scale()` function, which does just this.

```
# centre and then standardize
df_freq_full |>
  mutate(
         freq_z1 = (freq-mean(freq))/sd(freq),
         freq_z2 = scale(freq)) |>
  select(freq_z1, freq_z2) |>
  head()
```

```
# A tibble: 6 x 2
  freq_z1 freq_z2[,1]
    <dbl>       <dbl>
1 -0.0902     -0.0902
2 -0.0864     -0.0864
3 -0.0905     -0.0905
4 -0.0864     -0.0864
5 -0.0885     -0.0885
6 -0.0901     -0.0901
```

Let's use `scale()` for `freq` and `length`.

```
df_freq_full <-
  df_freq_full |>
  mutate(freq_z = scale(freq_log),
         length_z = scale(length))
```

```
fit_freq_z <-
  lm(rt ~ freq_z + length_z, data = df_freq_full)
```

First, let's check the R^2:

```
glance(fit_freq_z)$r.squared
```

```
[1] 0.4872977
```

We see that our R^2 value is 0.4872977, just like above. This serves as a reminder that the predictors still represent the same variance in the underlying model, their units and scales have simply changed. What about our coefficients:

```
tidy(fit_freq_z) |> select(term, estimate)
```

```
# A tibble: 3 x 2
  term        estimate
  <chr>          <dbl>
1 (Intercept)    770.
2 freq_z         -60.6
3 length_z        43.3
```

Here, a 1-unit change always corresponds to a change of 1 standard deviation. Now we see that frequency has a larger magnitude than the effect of length. So, for each instease in frequency by 1 standard deviation (holiding length constant), reaction times decrease by 29.5 ms.

4.2.1 Adding an interaction term

We won't spent much time talking about interactions, but please check out Ch. 8 (Interations and nonlinear effects) in Winter (2019) for a more in-depth treatment. For now, what's important to know is that interactions describe how effects of one predictor may be influenced by changes in another predictor. We can add interactin terms of two predictors by connecting them with a colon (`:`).

```
lm(rt ~ freq_z + length_z + freq_z:length_z,
   data = df_freq_full) |>
  tidy() |> select(term, estimate)
```

```
# A tibble: 4 x 2
  term            estimate
  <chr>              <dbl>
1 (Intercept)        766.
2 freq_z             -63.9
3 length_z            41.8
4 freq_z:length_z    -11.4
```

Or, we can simply connect the two predictors with an asterisk (`*`) to indicate that we want to look at both predictors and their interaction.

```
lm(rt ~ freq_z*length_z,
   data = df_freq_full) |>
  tidy() |> select(term, estimate)
```

```
# A tibble: 4 x 2
  term            estimate
  <chr>              <dbl>
1 (Intercept)        766.
2 freq_z             -63.9
3 length_z            41.8
4 freq_z:length_z    -11.4
```

The model estimates are the same for both models. The intercept is the predicted reaction time for a word with the mean length and mean frequency. Notice that the interaction slope is negative, meaning when both `freq` and `length` increase, reaction times will decrease.

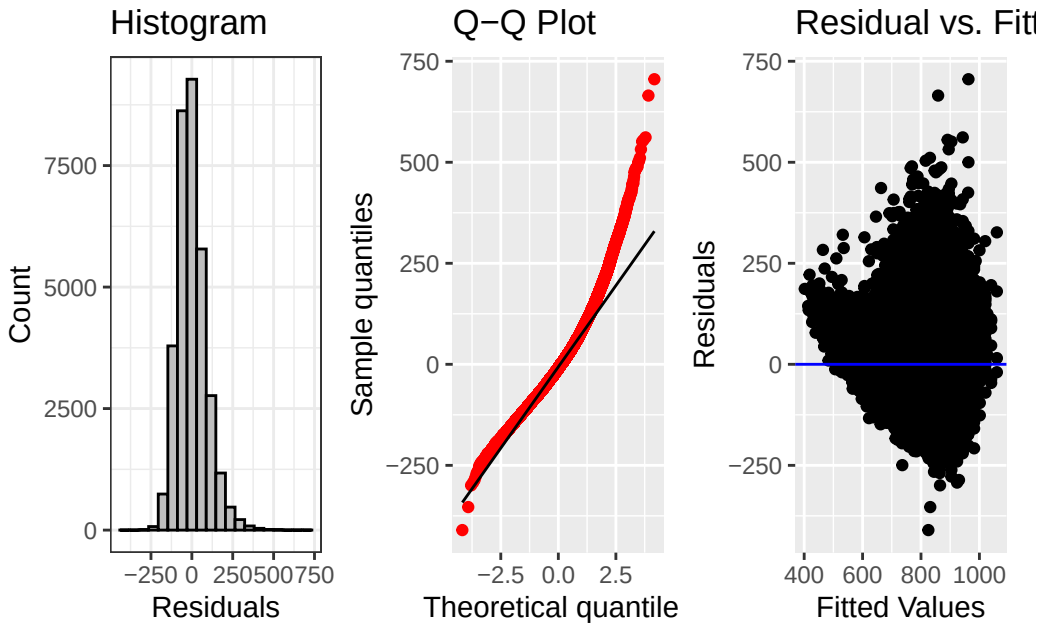
4.3 Model assumptions

We've already discussed the assumptions of normality and homoscedasticity (constant variance), which both refer to the residuals of a model. We typically assess these assumptions visually, with histogram and Q-Q plots.

4.3.1 Normality and Homoscedasticity

For our model

```
fig_hist <-
fit_freq_z |>
  ggplot() +
  aes(x = .resid) +
  geom_histogram(bins = 20, fill = "grey", colour = "black") +
  theme_bw() +
  labs(title='Histogram', x='Residuals', y='Count')

fig_qq <-
fit_freq_z |>
  ggplot() +
  aes(sample = .resid) +
  geom_qq(colour = "red") +
  geom_qq_line() +
  labs(title='Q-Q Plot', x='Theoretical quantiles', y='Sample quantiles')

fig_res <-
  fit_freq_z |>
  ggplot() +
  aes(x = .fitted, y = .resid) +
  geom_point() +
  geom_hline(yintercept = 0, colour = "blue") +
  labs(title='Residual vs. Fitted Values Plot', x='Fitted Values', y='Residuals')

fig_hist + fig_qq + fig_res
```



The histogram looks approximately normally distributed, with a bit of a positive skew. The Q-Q plot suggests a less-normal distribution, with the model estimates fitting larger reaction times more poorly. The residual plot also shows that the variance of the residuals is not constant, with much larger residual variance for larger fitted values. This tells us we should probably log reaction times. Let's try it all again, with log-transformed reaction times.

4.3.2 Log-transformed response variable

```
fit_freq_log_z <-
  lm(log(rt) ~ freq_z*length_z,
     data = df_freq_full)
```

```
glance(fit_freq_log_z)$r.squared
```

```
[1] 0.5176913
```

```
tidy(fit_freq_log_z) |> select(term, estimate)
```

```
# A tibble: 4 x 2
  term            estimate
```

```
  <chr>              <dbl>
1 (Intercept)      6.63
2 freq_z          -0.0826
3 length_z         0.0524
4 freq_z:length_z -0.00779
```

We see now that our values are much smaller, because they're on the log-scale.

```
  exp(6.63 + -0.0826*5 + 0.0524*2)
```

```
[1] 556.5739
```

```
  exp(6.63 + -0.0826*4 + 0.0524*2)
```

```
[1] 604.499
```

```
  exp(6.63 + -0.0826*1 + 0.0524*6)
```

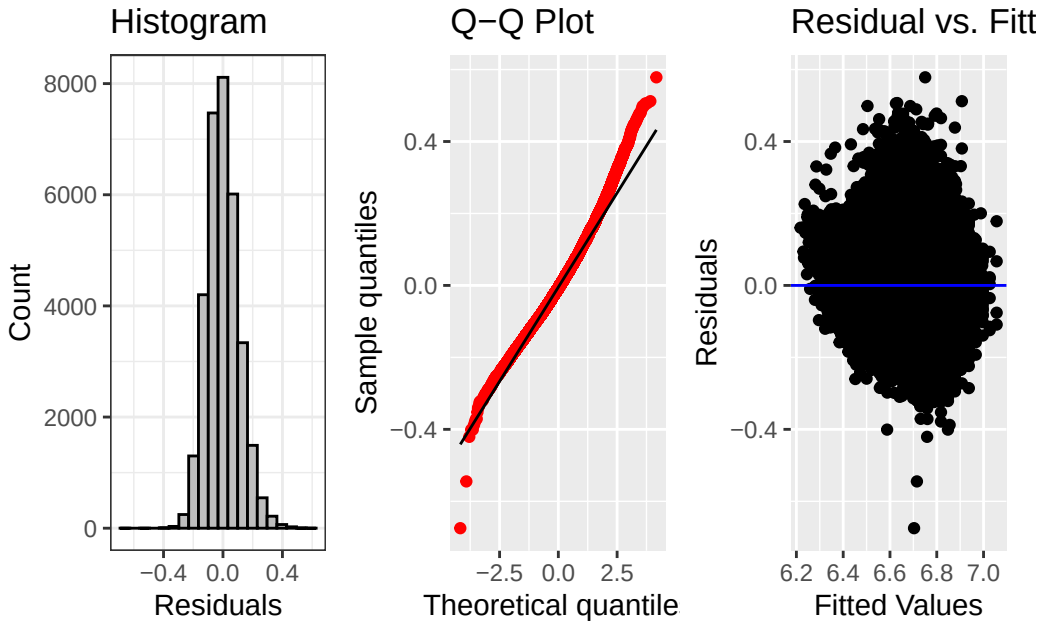
```
[1] 955.0847
```

```
  tidy(fit_freq_log_z)
```

```
# A tibble: 4 x 5
  term            estimate std.error statistic  p.value
  <chr>              <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)      6.63     0.000636   10428.  0
2 freq_z          -0.0826   0.000666    -124.  0
3 length_z         0.0524   0.000649      80.7 0
4 freq_z:length_z -0.00779  0.000581     -13.4 8.51e-41
```

```
  fig_hist <-
  fit_freq_log_z |>
    ggplot() +
    aes(x = .resid) +
    geom_histogram(bins = 20, fill = "grey", colour = "black") +
    theme_bw() +
```

```
  labs(title='Histogram', x='Residuals', y='Count')

fig_qq <-
fit_freq_log_z |>
  ggplot() +
  aes(sample = .resid) +
  geom_qq(colour = "red") +
  geom_qq_line() +
  labs(title='Q-Q Plot', x='Theoretical quantiles', y='Sample quantiles')

fig_res <-
  fit_freq_log_z |>
  ggplot() +
  aes(x = .fitted, y = .resid) +
  geom_point() +
  geom_hline(yintercept = 0, colour = "blue") +
  labs(title='Residual vs. Fitted Values Plot', x='Fitted Values', y='Residuals')

fig_hist + fig_qq + fig_res
```



Looks much better.

4.3.3 Collinearity

Collinearity refers to when continuous predictor variables are correlated, which can make the interpretation of their coefficients difficult, and the results spurious. Regression assumes there is an *absence* of collinearity, i.e., our predictor variables are not correlatded.

To assess collinearity, you can use the `vif()` function from the `car` package to compare *variance inflation factors*. VIF values close to 1 indicates there is not a high degree of collinearity between your variables.

```
car::vif(fit_freq_log_z)
```

```
     freq_z       length_z freq_z:length_z
   1.246509       1.184641       1.068283
```

Collinearity is a conceptual problem, and is something that you need to consider in the planning stage. Typically, we want to include predictors that we have specific predictions or research questions about. Shoving a bunch of predictors in a model to see what comes out significant is bad practice. Rather, we should have a principled approach to model building and variable selection. This is not to say that exploratory analyses should be avoided, but that this comes with caveats.

4.3.4 Adjusted R^2

Although we should avoid throwing any old predictor into our model, adjusted R^2 is a more conservative version of R^2 that takes into account the number of predictors in a model. For each additional predictor, adjusted R^2 includes the number of predictors (k) in its denominator (bottom half of a division), which means that the more predictors there are, the smaller R^2 will be, unless each additional predictor explains sufficient variance to counteract this penalisation.

```
glance(fit_freq_log_z)$adj.r.squared
```

```
[1] 0.5176475
```

If we were to look at the (adjusted) R^2 of our simple linear regression model, where log reaction times are predicted by standardised log frequency, we see that there is a large increase in our model which includes length and its interaction. This suggests that our model is not overfit, and that length contributes to the variance explained by the model.

```r
glance(lm(log(rt) ~ freq_z, data = df_freq_full))$adj.r.squared
```

```
[1] 0.4148675
```

If we likewise compare to the same model without an interaction term (log reaction times ~ frequency * length), we see that the adjusted R^2 is not very different. If the adjusted R^2 were much lower, this would indicate that including the interaction term leads to overfitting.

```r
glance(lm(log(rt) ~ freq_z + length_z, data = df_freq_full))$adj.r.squared
```

```
[1] 0.5150461
```

Important terms

Term	Definition	Equation/Code
Collinearity	Correlation between two predictors (linear model assumes non-collinearity)	car::vif(model)
NA	NA	NA

Learning Objectives

Today we learned...

Today we will learn...

- what multiple regression is
- how to include multiple predictor variables
- how to interpret slopes in multiple regression
- how to interpret interaction effects
- about the assumption of the absence of collinearity

4.4 Task

Load in the `english` dataset from the `languageR` package (Baayen & Shafaei-Bajestan, 2019) (code below). You don't need to load in any CSV file, because this dataset is available if you have the package loaded. From the manual:

> This data set gives mean visual lexical decision latencies and word naming latencies to 2284 monomorphemic English nouns and verbs, averaged for old and young subjects, with various predictor variables.

(languageR manual, p. 29)

```
# load in 'english' dataset from languageR
df_freq_eng <-
  as.data.frame(english) |>
  dplyr::select(RTlexdec, RTnaming, Word, LengthInLetters, AgeSubject, WrittenFrequency) |
  rename(rt_lexdec = RTlexdec,
         rt_naming = RTnaming,
         freq_written = WrittenFrequency) |>
  clean_names() |>
  relocate(word)
```

We're keeping five variables:

- `word`: a factor with 2284 words
- `rt_lexdec`: numeric vector of log RT in visual lexical decision
- `rt_naming`: numeric vector of log RT in word naming
- `length_in_letters`: numeric vector with length of the word in letters
- `AgeSubject`: a factor with as levels the age group of the subject: young versus old.
- `freq_written`: numeric vector with log frequency in the CELEX lexical database

Take the following steps:

1. Perform an exploratory data analysis to understand the data (produce plots, tables, whatever you think necessary and can do).
2. Model the data, with *back-transformed* (raw) reaction times as a response variable and written frequency and length in letters as predictors. Perform any tranformations you think necessary. Run model diagnostic checks and assess model fit.
3. Re-run the model with log reaction times as a response variable and written frequency and length in letters as predictors. Perform any tranformations you think necessary. Run model diagnostic checks and assess model fit.
4. Remove length in letters as a predictor. How is model fit affected? What can you conclude?

5 Categorical predictors

Regression for Linguists

Learning Objectives

Today we will learn...

- about cateogorical predictors
- how to interpret different contrast coding

Set-up environment

```
# suppress scientific notation
options(scipen=999)
```

We'll also need to load in our required packages.

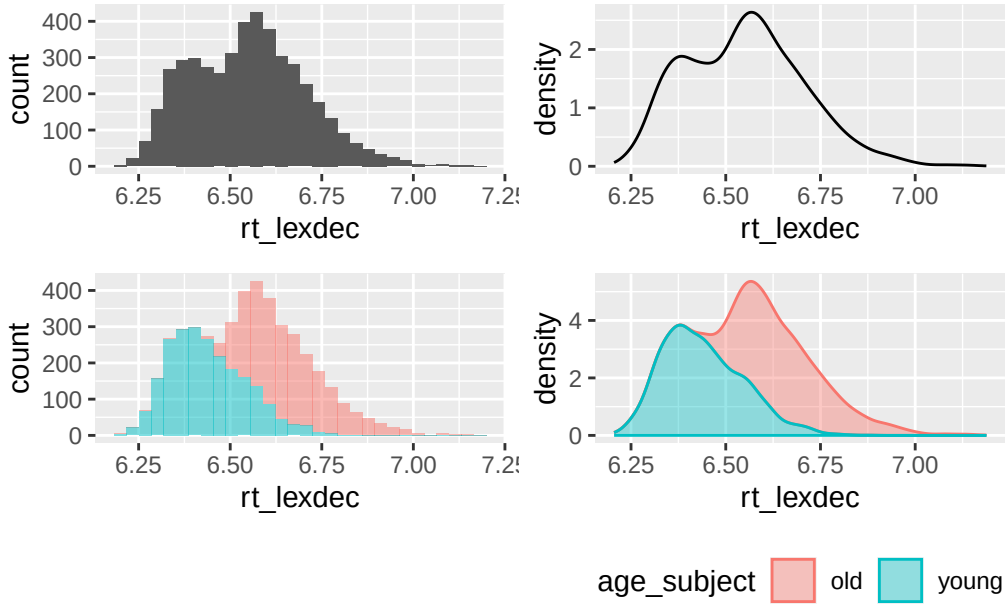
```
# load libraries
pacman::p_load(
               tidyverse,
               here,
               broom,
               lme4,
               janitor,
               languageR)
```

Load data

Let's continue working with the `english` dataset from the `languageR` package. Let's just call it `df_freq_eng`.

```
df_freq_eng <-
  as.data.frame(english) |>
  dplyr::select(RTlexdec, RTnaming, Word, LengthInLetters, AgeSubject, WrittenFrequency) |
  rename(rt_lexdec = RTlexdec,
         rt_naming = RTnaming,
         freq_written = WrittenFrequency) |>
  clean_names() |>
  # standardize continuous predictors
  mutate(
    freq_z = scale(freq_written),
    length_z = scale(length_in_letters)
  ) |>
  relocate(word) |>
  arrange(word)
```

In your exploratory data analysis, you might've noticed a *bimodal* distribution.



This looks like a *bimodal* distribution, i.e., there are two *modes* (most frequent value, i.e., peak in a histogram). What might be driving this? We know that there were two subject groups: old and young. How does the distribution of these two groups look?

Running our model of the log reaction times as predicted by frequency and length, we see:

```
fit_freq_length <-
  lm(rt_lexdec ~ freq_z*length_z,
     data = df_freq_eng)
```

```
glance(fit_freq_length)$r.squared
```

```
[1] 0.1896649
```

```
glance(fit_freq_length)$adj.r.squared
```

```
[1] 0.1891323
```

Seems like we don't have any overfitting in our model (R^2 and adjusted R^2 are comparable). Let's look at our coeffiecients.

```
tidy(fit_freq_length) |> select(term, estimate)
```

```
# A tibble: 4 x 2
  term            estimate
  <chr>              <dbl>
1 (Intercept)      6.55
2 freq_z          -0.0682
3 length_z         0.00328
4 freq_z:length_z -0.00196
```

There is a negative slope for frequency, indicating shorter reaction times for words with higher frequency (when holding length constant). There is a positive slope for length, indicating longer reaction times for longer words (holding frequency constant). There is also a negative interaction estimate, indicating that when both length and frequency increase, reaction times decrease. This seems similar to the dataset we explored in the previous sections. But, this bimodal distribution is suggesting we should include age group as a predictor, since the two groups seem to pattern differently in their reading times. Could it be that the effect of frequency and length also differ as a function of age group?

5.1 Categorical predictors

In linguistic research we often want to compare the effect of *groups* or categories, such as native or non-native speakers, or grammatical or ungrammatical stimuli. We might expect longer reading times for non-native (compared to native) speakers of a language, or for ungrammatical (versus grammatical) sentences. With our current dataset, we'd predict longer reading times for older participants than younger participants (although we should hypothesise *before* collecting and visualising our data!). How might these age effects interact with effects of word frequency and length?

5.1.1 Including a categorical predictor

What would happen if we just include `age_subject` in our model?

```
fit_age <-
  lm(rt_lexdec ~ freq_z*length_z + age_subject,
     data = df_freq_eng)
```

First, we see that adding age to our model results in a large increase in variance explained, and that the R^2 and adjusted R^2 values are comparable. In addition, the VIF values for all coefficients are near 1. This indicates that our predictors all contribute to the variance explained by the model and are not correlated.

```
glance(fit_age)$r.squared
```

```
[1] 0.6888949
```

```
glance(fit_age)$adj.r.squared
```

```
[1] 0.6886222
```

```
car::vif(fit_age)
```

```
      freq_z     length_z  age_subject freq_z:length_z
    1.012553     1.004461     1.000000     1.008108
```

Now that we see that our model is not overfit and that our predictors are not correlatd, let's take a look at our model estimates.

```
tidy(fit_age) |> select(term,estimate)
```

```
# A tibble: 5 x 2
  term              estimate
  <chr>                <dbl>
1 (Intercept)       6.66
2 freq_z           -0.0682
3 length_z          0.00328
4 age_subjectyoung -0.222
5 freq_z:length_z  -0.00196
```

In addition to the effects we observed in our earlier model, we see that there is a negative slope for `age_subjectyoung`, indicating that reaction times decrease when…what? How do we interpret a slope for a categorical variable? Regression works with numerical values, so how does a categorical variable get fit to a line? If we feed a categorical variable into the `lm()` function, the factor levels (i.e., the categories in a categorical variable) are given numerical values. We need to know what these values are in order to know how to interpret our model estimates. We call these numerical values mapped onto factor levels contrast coding, and we can check the contrasts of a given factor using the function `contrasts()`.

```
contrasts(df_freq_eng$age_subject)
```

```
      young
old       0
young     1
```

We see that `old` was coded at 0 and `young` as 1. This means that our slope for `age_subjectyoung` represents the change in reaction times when we move from `old` to `young`, which corresponds to a 1-unit change in our predictor (because the difference between 0 and 1 is 1). This is called treatment coding, or dummy coding, where one factor level is coded as 0 and the other as 1. Let's remove the continuous variable for now and focus on `age_subject`. Let's also look at raw reaction times, to more easily interpret the results.

```
fit_age <-
  lm(exp(rt_lexdec) ~ age_subject,
     data = df_freq_eng)
```

```
glance(fit_age)$r.squared
```

```
[1] 0.4682224
```

Our R^2 value is lower than when we included frequency and length, but higher still than our model with frequeny and length but no age.

```
tidy(fit_age) |> select(term, estimate)
```

```
# A tibble: 2 x 2
  term              estimate
  <chr>                <dbl>
1 (Intercept)          787.
2 age_subjectyoung    -157.
```

We see that there is an estimated decrease in reaction times of 157ms for the young group compared to the old group. But what does the intercept represent here? Let's look at our data again.

```
df_freq_eng |>
  select(rt_lexdec, age_subject) |>
  mutate(rt_lexdec = exp(rt_lexdec)) |>
  summary()
```

```
   rt_lexdec      age_subject
 Min.   : 495.4   old  :2284
 1st Qu.: 617.4   young:2284
 Median : 699.6
 Mean   : 708.1
 3rd Qu.: 775.3
 Max.   :1323.2
```

And how does `rt_lexdec` differ between the groups?

```
df_freq_eng |>
  select(rt_lexdec, age_subject) |>
  mutate(rt_lexdec = exp(rt_lexdec)) |>
  summarise(mean = mean(rt_lexdec),
            min = min(rt_lexdec),
            max = max(rt_lexdec),
    .by = "age_subject"
  )
```

```
  age_subject     mean    min    max
1       young 629.5473 495.38  971.8
2         old 786.7200 603.77 1323.2
```

We see here that the intercept for our model actually corresponds to the mean reaction time for the old group. Why is this? Recall that the intercept corresponds to the y value (reaction time) when x is 0. In treatment/dummy coding, one factor level is coded as 0. In our case this was `old`, and so the intercept corresponds to the mean reaction time for participants in the old group. How does R choose which variable to code as 0? It simply takes the first level name alphabetically: `old` comes before `young`, so `old` was automatically taken as the 'baseline' to which `young` was compared.

And if we were to add the slope to the intercept, we would get the mean for the *young* group. Why is this?

```
coef(fit_age)['(Intercept)'] + coef(fit_age)['age_subjectyoung']
```

```
(Intercept)
   629.5473
```

Why are the means for the two groups used? The mean is the value closest to all values in a univariate dataset, and regression aims to inimise residuals (recall the line of best fit). So, a line is fit between the means of these two factor levels to achieve minimal residuals. This actually is the same thing as a t-test:

```
t.test(exp(rt_lexdec) ~ age_subject, data = df_freq_eng)
```

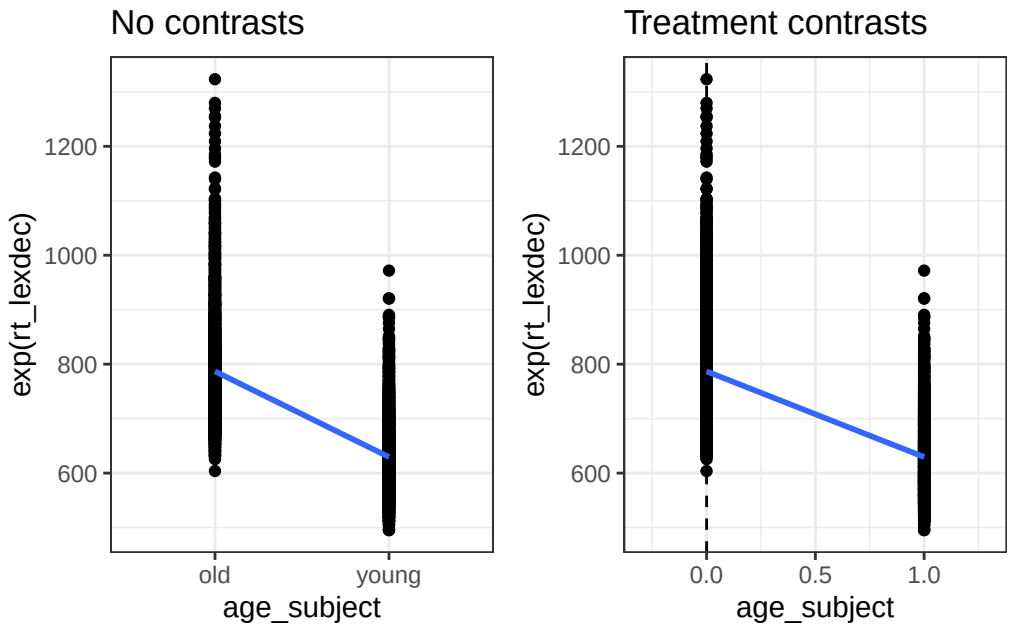
```
    Welch Two Sample t-test

data:  exp(rt_lexdec) by age_subject
t = 63.406, df = 4144.6, p-value < 0.00000000000000022
alternative hypothesis: true difference in means between group old and group young is not eq
95 percent confidence interval:
 152.3128 162.0325
sample estimates:
  mean in group old mean in group young
           786.7200            629.5473
```

If we compare this to our model, we see that the t- and p-values are identical (more on these later).

```r
tidy(fit_age)
```

```
# A tibble: 2 x 5
  term             estimate std.error statistic p.value
  <chr>               <dbl>     <dbl>     <dbl>   <dbl>
1 (Intercept)          787.      1.75     449.        0
2 age_subjectyoung    -157.      2.48     -63.4       0
```

```r
fig_nocontrasts <-
df_freq_eng |>
  ggplot() +
  aes(x = age_subject, y = exp(rt_lexdec)) +
  labs(title = "No contrasts") +
  # geom_vline(xintercept = 0, linetype="dashed", size = .5) +
  geom_point(position = position_dodge(.6)) +
  geom_smooth(method = 'lm', aes(group=1)) + theme_minimal() +
  theme_bw()

fig_treatment <-
df_freq_eng |>
  mutate(age_subject = if_else(age_subject=="young",1,0)) |>
  ggplot() +
  aes(x = age_subject, y = exp(rt_lexdec)) +
  labs(title = "Treatment contrasts") +
  geom_vline(xintercept = 0, linetype="dashed", size = .5) +
  geom_point(position = position_dodge(.6)) +
  geom_smooth(method = 'lm', aes(group=1)) + theme_minimal() +
  theme_bw()

fig_nocontrasts + fig_treatment
```



5.2 Sum contrasts

Treatment/dummy coding is the default contrast coding scheme. Sum coding is another frequently used coding scheme, which is essentially centring categorical variables. Just as with continuous variables, the motivation for sum contrast coding mainly lies in the interpretation of interaction effects. How can we tell R we want to use sum contrast coding, and not dummy coding? There are different ways to do this:

```
# first, make sure your variable is a factor
df_freq_eng$age_subject <- as.factor(df_freq_eng$age_subject)
# check
class(df_freq_eng$age_subject)
```

```
[1] "factor"
```

```
# next, you could use the contr.sum() function
contrasts(df_freq_eng$age_subject) <- contr.sum(2) # where 2 means we have 2 levels
contrasts(df_freq_eng$age_subject)
```

```
      [,1]
old      1
young   -1
```

Here we see that `old` is coded as −1 and `young` as +1. I prefer to use +/-0.5 for reasons we don't need to go into here. I would also prefer to have `young` coded in the negative value, and `old` in the positive value. This aids in the way I interpret the slope: a change in reaction times for the older group compared to the younger group.

```
#or, you could manually control the sum contrasts
## check the order of the levels
levels(df_freq_eng$age_subject)
```

```
[1] "old"   "young"
```

```
## code 'old' as +.5 and 'young' as -.5
contrasts(df_freq_eng$age_subject) <- c(+0.5, -0.5)
contrasts(df_freq_eng$age_subject)
```

```
      [,1]
old    0.5
young -0.5
```

You could also choose to store the contrast values in their own variable.

```
df_freq_eng <-
  df_freq_eng |>
  mutate(age_numeric = ifelse(age_subject == "young", -0.5, +0.5))
```

```
df_freq_eng |>
  select(age_subject, age_numeric) |>
  head()
```

```
     age_subject age_numeric
338        young        -0.5
1790         old         0.5
3125       young        -0.5
3957         old         0.5
3313       young        -0.5
4145         old         0.5
```

Now, we can run our model using either `age_subject` or `age_numeric`.

```
fit_age_sum <-
  lm(exp(rt_lexdec) ~ age_subject,
     data = df_freq_eng)
```

```
glance(fit_age_sum)$r.squared
```

```
[1] 0.4682224
```

```
glance(fit_age)$r.squared
```

```
[1] 0.4682224
```

No difference in variance account for by our model.

```
tidy(fit_age_sum) |> select(term,estimate)
```

```
# A tibble: 2 x 2
  term         estimate
  <chr>           <dbl>
1 (Intercept)      708.
2 age_subject1     157.
```

But there is a difference in the intercept, and a change in sign in our slope. Why is this?

```
fig_sum1 <-
df_freq_eng |>
  mutate(age_subject = if_else(age_subject=="young",-1,1)) |>
  ggplot() +
  aes(x = age_subject, y = exp(rt_lexdec)) +
  labs(title = "Sum contrasts") +
  geom_vline(xintercept = 0, linetype="dashed", size = .5) +
  geom_point(position = position_dodge(.6)) +
  geom_smooth(method = 'lm', aes(group=1)) + theme_minimal() +
  theme_bw()

fig_sum5 <-
df_freq_eng |>
  mutate(age_subject = if_else(age_subject=="young",-.5,.5)) |>
```

```
  ggplot() +
  aes(x = age_subject, y = exp(rt_lexdec)) +
  labs(title = "Sum contrasts") +
  geom_vline(xintercept = 0, linetype="dashed", size = .5) +
  geom_point(position = position_dodge(.6)) +
  geom_smooth(method = 'lm', aes(group=1)) + theme_minimal() +
  theme_bw()

fig_treatment + fig_sum5 + plot_annotation(tag_levels = "A")
```

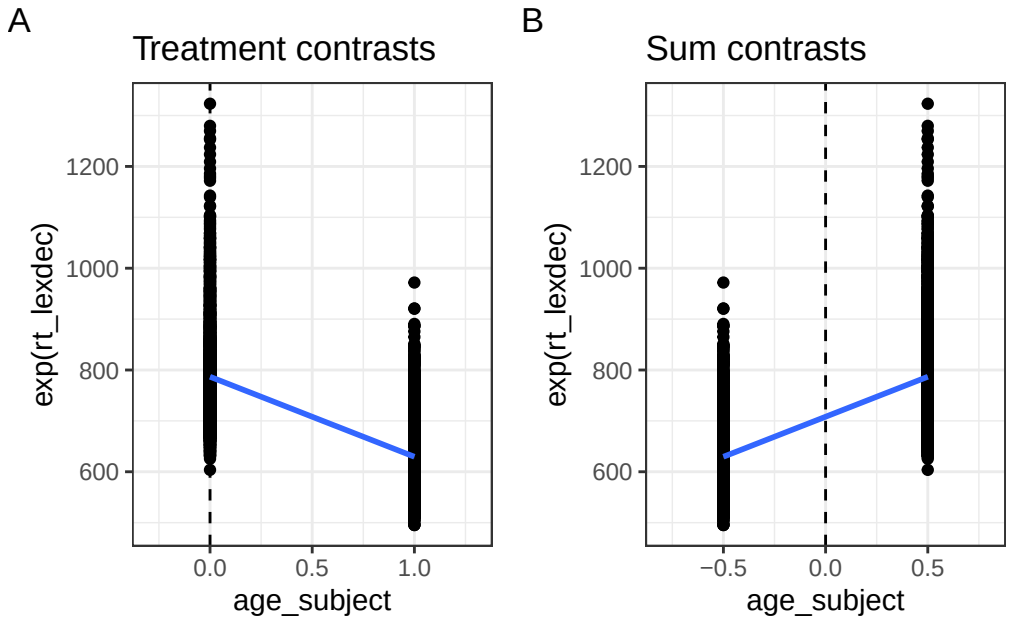


Figure 5.1: The difference in slope corresponds to which level is coded as 0 (dummy coding) or -5/-1 (sum coding)

As we see in Figure 5.1, the sign of the slope depends on how we've contrast coded our factor levels. In Figure 5.1 A, the `old` group is coded as 0 and `young` as 1. In Figure 5.1 B, the `young` group is coded as $-.5$ and the `old` group as $+.5$.

The intercept value is also now the overall mean of all observed reaction times, because now the y value when x equals zero lies in the middle of the two groups. The slope magnitude (i.e., size of the value) hasn't changed, because the difference betwen the two group means has not changed.

```
mean(exp(df_freq_eng$rt_lexdec))
```

```
[1] 708.1336
```

5.2.1 Exploring predicted values

Let's also explore the predicted values of our model with a categorical variable.

```
head(fitted(fit_age), n = 10)
```

```
     338      1790      3125      3957      3313      4145       337      1789
629.5473 786.7200 629.5473 786.7200 629.5473 786.7200 629.5473 786.7200
    3513      4345
629.5473 786.7200
```

We see that there are only 2 values, 630 and 787. These correspond to the means for each group that we saw above. They also seem to be in a pattern: young-mean, old-mean, young-mean, old-mean, etc. How does this correspond to the age group of the participant for the first ten observations?

```
head(df_freq_eng$age_subject, n = 10)
```

```
 [1] young old   young old   young old   young old   young old
attr(,"contrasts")
      [,1]
old    0.5
young -0.5
Levels: old young
```

The first ten observations in our data are in young-old pairs. What are the first values in the raw data?

```
head(exp(df_freq_eng$rt_lexdec), n = 10)
```

```
 [1] 623.61 775.67 617.10 715.52 575.70 742.19 592.42 748.37 541.67 824.76
```

And what is the difference between these reaction times and the fitted values?

```
head(exp(df_freq_eng$rt_lexdec), n = 10) - head(fitted(fit_age), n = 10)
```

term	description/other terms

```
      338       1790       3125       3957       3313       4145        337
-5.937299 -11.049991 -12.447299 -71.199991 -53.847299 -44.529991 -37.127299
     1789       3513       4345
-38.349991 -87.877299  38.040009
```

```
head(residuals(fit_age))
```

```
      338       1790       3125       3957       3313       4145
-5.937299 -11.049991 -12.447299 -71.199991 -53.847299 -44.529991
```

5.3 Summary

- we saw that the equation for a straight line boils down to its intercept and slope
- we fit our first linear model with a categorical predictor

Important terms

Learning Objectives

Today we learned...

5.4 Task

We'll use a dataset from Biondo et al. (2022), an eye-tracking reading study exploring the processing of adverb-tense concord in Spanish past and future tenses. Participants read sentences that began with a temporal adverb (e.g., yesterday/tomorrow), and had a verb marked with the congruent or incongruent tense (past/future).

Load in the data.

```
df_tense <-
  read_csv(here("data", "Biondo.Soilemezidi.Mancini_dataset_ET.csv"),
           locale = locale(encoding = "Latin1") # for special characters in Spanish
           ) |>
  mutate(gramm = ifelse(gramm == "0", "ungramm", "gramm")) |>
```

```
  clean_names()
```

5.4.1 Treatment contrasts

We will look at the measure *total reading time* (`tt`) at the *verb* region (`roi == 4`). Subset the data to only include the verb region.

```
df_verb <-
  df_tense |>
  filter(roi == 4)
```

1. Run a simple linear model with (log-transformed) total reading time (`tt`) as an independent variable and grammaticality (`gramm`) as a dependent variable. Use treatment contrasts.
2. Inspect your coefficients again. What conclusions do you draw?
3. Run model diagnostics:
    - check model assumptions where relevant (normality, constant variance, collinearity)
    - check model fit (R^2)

5.4.2 Sum contrasts

1. Re-run your model with sum contrasts.
2. Inspect your coefficients again. Do your conclusions change?
3. Re-run your model diagnostics. How does it compare to your first model?

5.4.3 Multiple regression

1. Add verb tense (`verb_t`: past, future) as a predictor, including an interaction term. Use sum contrasts.
2. Inspect your coefficients again. Do your conclusions change?
3. Re-run your model diagnostics. How does it compare to the last models?

Literaturverzeichnis

Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics using R*.

Baayen, R. H., Davidson, D. J., & Bates, D. M. (2008). Mixed-effects modeling with crossed random effects for subjects and items. *Journal of Memory and Language*, *59*(4), 390–412. https://doi.org/10.1016/j.jml.2007.12.005

Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing linguistic data: A practical introduction to statistics*. https://CRAN.R-project.org/package=languageR

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language*, *68*(3), 255–278. https://doi.org/10.1016/j.jml.2012.11.001

Bates, D., Kliegl, R., Vasishth, S., & Baayen, H. (2015). Parsimonious Mixed Models. *arXiv Preprint*, 1–27. https://doi.org/10.48550/arXiv.1506.04967

Biondo, N., Soilemezidi, M., & Mancini, S. (2022). Yesterday is history, tomorrow is a mystery: An eye-tracking investigation of the processing of past and future time reference during sentence reading. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *48*(7), 1001–1018. https://doi.org/10.1037/xlm0001053

Jaeger, T. F. (2008). Categorical data analysis: Away from ANOVAs (transformation or not) and towards logit mixed models. *Journal of Memory and Language*, *59*(4), 434–446. https://doi.org/10.1016/j.jml.2007.11.007

Lüdecke, D., Ben-Shachar, M. S., Patil, I., Waggoner, P., & Makowski, D. (2021). performance: An R package for assessment, comparison and testing of statistical models. *Journal of Open Source Software*, *6*(60), 3139. https://doi.org/10.21105/joss.03139

Matuschek, H., Kliegl, R., Vasishth, S., Baayen, H., & Bates, D. (2017). Balancing Type I error and power in linear mixed models. *Journal of Memory and Language*, *94*, 305–315. https://doi.org/10.1016/j.jml.2017.01.001

Sonderegger, M. (n.d.). *Regression Modeling for Linguistic Data*.

Sonderegger, M. (2023). *Regression Modeling for Linguistic Data*.

Vasishth, S. (2022). *Some right ways to analyze (psycho)linguistic data* [Preprint]. PsyArXiv. https://doi.org/10.31234/osf.io/y54va

Vasishth, S., & Nicenboim, B. (2016). Statistical methods for linguistic research: Foundational Ideas—Part I. *Language and Linguistics Compass*, *10*(11), 591–613. https://doi.org/10.1111/lnc3.12207

Winter, B. (2013). *Linear models and linear mixed effects models in R: Tutorial 1*.

Winter, B. (2014). *A very basic tutorial for performing linear mixed effects analyses (Tutorial 2)*.

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

Part IV

Day 3: Logistic regression

6 Logistic regression

Regression for Linguists

This lecture jumps to chapter 12 'Genearlised Lienar Models 1: Logistic Regression' (Winter, 2019). We're skipping a few chapters, which I encourage you to go through on your own. However, they cover topics that you presumably have covered in previous courses (namely significance testing, t-values and p-values).

Learning Objectives

Today we will learn...

- how to model binomial data with logistic regression
- how to interpret log-odds and odds ratio

Set-up environment

```
# suppress scientific notation
options(scipen=999)
options(pillar.sigfig = 5)
```

```
library(broman)
# function to format p-values
format_pval <- function(x){
  if (x < .001) return(paste('<', '.001'))
  if (x < .01) return(paste('<', '.01'))
  if (x < .05) return(paste('<', '.05'))
  paste('=', myround(x, 3))  # if above .05, print p-value to 3 decimalp points
}
```

We'll also need to load in our required packages. Hopefully you've already install the required packages (if not, go to **?@sec-software**).

```
# load libraries
pacman::p_load(
               tidyverse,
               here,
               broom,
               lme4,
               janitor,
               languageR)
```

```
# set preferred ggplot2 theme
theme_set(theme_bw() + theme(plot.title = element_text(size = 10)))
```

6.1 Generalised linear models

Logistic regression is a type of genearlised linear model (GLM), and is used to model binomial response data. Whereas continuous response variables, such as reaction times, assume a normal distribution (a.k.a., a Gaussian distribution), logistic regression assumes a binomial distribution (a.k.a., Bernoulli distribution). These are formalised in equations 6.1, where μ and σ correspond to the mean and standard deviation, and 6.2, where N and p refer to the number of trials and the probability of y being 1 or 0.

$$y \sim Normal(\mu, \sigma) \tag{6.1}$$

$$y \sim binomial(N = 1, p) \tag{6.2}$$

Don't stress about this for now, I find the math behind everything will start to make more sense the more often you see it. However, *some* math is necessary in order to understand the output of our models, namely the relation between probabilities, odds, and log odds.

6.1.1 Log-odds, odds ratio, and probabilities

In logistic regression, we the probability (p) of observing one outcome or another as a function of a predictor variable. In linguistic research, these outcomes could be the absence or presence of some phenomenon (pause, schwa, etc.) or button responses (yes/no, accept/reject). In logistic regression, we describe the probability, odds, or log-odds of a particular outcome over another.

Table 6.1: Comparison of different values of probabilities/odds/log-odds

name	1	2	3	4	5	6	7	8
prob	0.0066929	0.0229774	0.0758582	0.2227001	0.5	0.7772999	0.9241418	0.9770226
odds	0.0067379	0.0235177	0.0820850	0.2865048	1.0	3.4903430	12.1824940	42.5210820
log_odds	-5.0000000	-3.7500000	-2.5000000	-1.2500000	0.0	1.2500000	2.5000000	3.7500000

Probability is quite intuitive, and ranges from 0 (no chance) to 1 (certain). A 50% chance corresponds to a probability of 0.5. You're also likely familiar with odds, which can range from 0 to infinity. Odds are often used in betting, such as *the odds that I'll win are 2:1*, which corresponds to $\frac{2}{1} = 2$ in favour of my winning. Conversely, *the odds that you'll win are 1:2*, corresponding to $\frac{1}{2} = 0.5$, meaning it's less likely that you'll win compared to you losing. If the odds are even, then: $\frac{1}{1} = 1$. So, odds of 1 correspond to a probability of 0.5. Log-odds are just the logarithmically-transformed odds: $log(2) = 0.6931472$; $log(0.5) = -0.6931472$; $log(1) = 0$. Probability can also be computed using the odds, as shown in 6.3: $\frac{2}{1+2} = 0.6666667$; $\frac{1}{1+1} = 0.5$; $\frac{0.5}{1+0.5} = 0.3333333$.

We can get the probability from a log odds value using `plogis()`, which performs the following calculation:

$$p = \frac{exp(log\ odds)}{1 + exp(log\ odds)} = \frac{odds}{1 + odds} \tag{6.3}$$

Table 6.1 gives an example of how the three relate to each other. The grey cells are all where chances re 50/50, with increasingly more likely (green) or less likely (red) values/

This relationship is demonstrated in Figure 6.1. Take your time to really understand these plots, as it will help understand the output of our models.

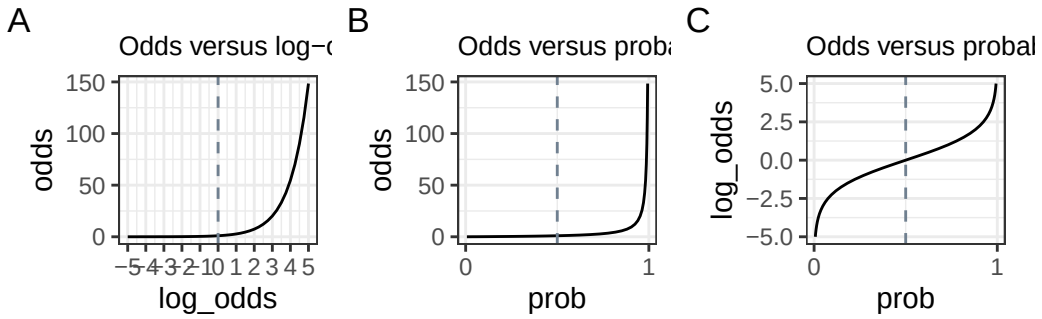


Figure 6.1: Relationship between probability, odds, and log-odds

6.2 Logistic regression

I find the more we talk about the math behind the models before even running a model, the more overwhelmed we become. So, let's run our first logistic regression and then dissect it to understand it. Most relevant to the output of a logistic regression model is Figure 6.1 C, as the model will output log-odds, and we most likely want to interpret them in terms of probabilities.

We'll use a dataset from Biondo et al. (2022), an eye-tracking reading study exploring the processing of adverb-tense concord in Spanish past and future tenses. Participants read sentences that began with a temporal adverb (e.g., yesterday/tomorrow), and had a verb marked with the congruent or incongruent tense (past/future). We will look at the measure *regression in* at the *verb* region.

Let's start by loading in the data:

```
df_tense <-
  read_csv(here("data", "Biondo.Soilemezidi.Mancini_dataset_ET.csv"),
           locale = locale(encoding = "Latin1") # for special characters in Spanish
           ) |>
  mutate(gramm = ifelse(gramm == "0", "ungramm", "gramm")) |>
  clean_names() |>
  filter(roi == 4,
         adv_type == "Deic")
```

6.2.1 EDA

And conducting a quick EDA: print summaries and plot the response variables.

```
head(df_tense)
```

```
# A tibble: 6 x 13
  sj     item adv_type adv_t  verb_t gramm     roi label    fp    gp    tt    ri
  <chr> <dbl> <chr>    <chr>  <chr>  <chr>   <dbl> <chr> <dbl> <dbl> <dbl> <dbl>
1 1        54 Deic     Past   Past   gramm       4 enca~  1027  1027  1027     0
2 1         4 Deic     Future Future gramm       4 cole~   562   562  1337     1
3 1        62 Deic     Past   Past   gramm       4 esqu~   293  1664  1141     0
4 1        96 Deic     Future Past   ungramm     4 cons~   713  1963  1868     0
5 1        52 Deic     Past   Past   gramm       4 desa~   890   890  1707     1
6 1        90 Deic     Future Past   ungramm     4 dece~   962   962   962     0
# i 1 more variable: ro <dbl>
```

Let's look at only our binomial dependent variables, regression in (`ri`) and regression out (`ro`). For each variable, `1` indicates a regression in/out, `0` indicates there was no regression in/out.

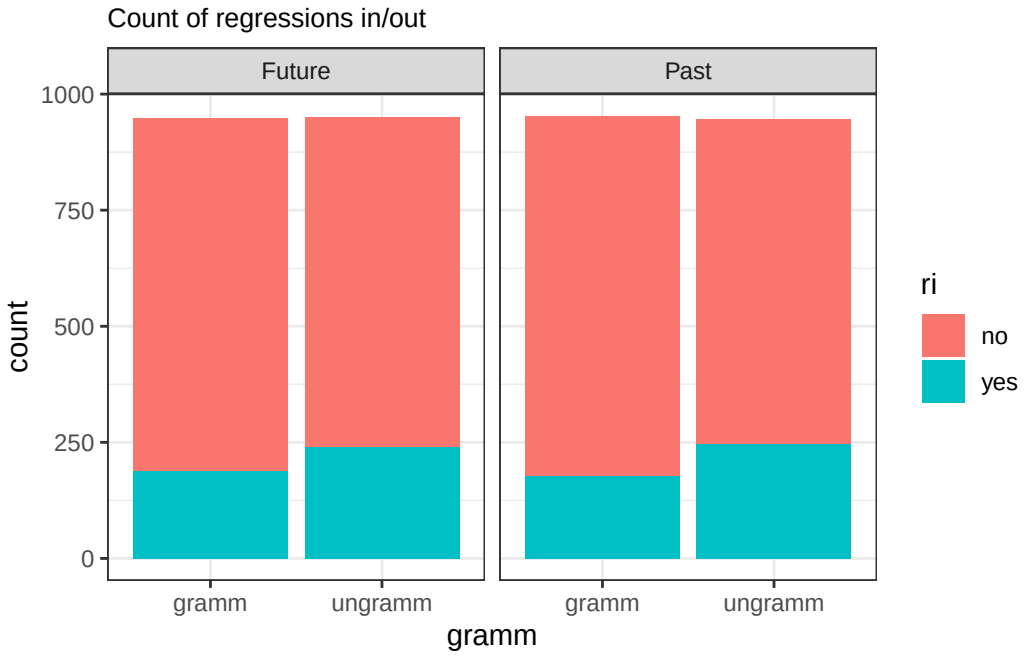
```
df_tense |>
  select(roi, ri, ro) |>
  summary()
```

```
      roi          ri               ro          
 Min.   :4   Min.   :0.0000   Min.   :0.00000  
 1st Qu.:4   1st Qu.:0.0000   1st Qu.:0.00000  
 Median :4   Median :0.0000   Median :0.00000  
 Mean   :4   Mean   :0.2248   Mean   :0.08169  
 3rd Qu.:4   3rd Qu.:0.0000   3rd Qu.:0.00000  
 Max.   :4   Max.   :1.0000   Max.   :1.00000  
             NA's   :45       NA's   :45       
```

Let's plot out our dependent variable of interest: regression in.

```
# make grammaticality a factor
df_tense |>
  mutate(gramm = as_factor(gramm))
```

```
# A tibble: 3,840 x 13
   sj     item adv_type adv_t  verb_t gramm    roi label    fp    gp    tt    ri
   <chr> <dbl> <chr>    <chr>  <chr>  <fct>  <dbl> <chr> <dbl> <dbl> <dbl> <dbl>
 1 1        54 Deic     Past   Past   gramm      4 enca~  1027  1027  1027     0
 2 1         4 Deic     Future Future gramm      4 cole~   562   562  1337     1
 3 1        62 Deic     Past   Past   gramm      4 esqu~   293  1664  1141     0
 4 1        96 Deic     Future Past   ungra~     4 cons~   713  1963  1868     0
 5 1        52 Deic     Past   Past   gramm      4 desa~   890   890  1707     1
 6 1        90 Deic     Future Past   ungra~     4 dece~   962   962   962     0
 7 1         8 Deic     Future Future gramm      4 evid~   718   718   718     0
 8 1         9 Deic     Future Future gramm      4 excu~  1453  1453  1453     0
 9 1        56 Deic     Past   Past   gramm      4 equi~   769   769   769     0
10 1        11 Deic     Future Future gramm      4 cena~   778   778   778     0
# i 3,830 more rows
# i 1 more variable: ro <dbl>
```



It looks like there are more regressions in for the grammatical versus ungrammatical conditions in both the future and past tenses. There doesn't seem to be a large difference between the two tenses in overall regressions in, nor in the effect of grammaticality on the proportion of regressions in. We can also see that it was more likely overall for there to *not* be a regression in (versus for there to be a regression in).

6.2.2 Model

Now let's run our model. Verb tense and grammaticality are each two-level factors, so we'll want to set sum coding for each of them. Let's set `past` and `grammatical` to −0.5, and `future` and `ungrammatical` to `+0.5`.

```
df_tense$verb_t <- as.factor(df_tense$verb_t)
levels(df_tense$verb_t)
```

```
[1] "Future" "Past"
```

```
contrasts(df_tense$verb_t) <- c(+0.5, -0.5)
contrasts(df_tense$verb_t)
```

```
       [,1]
```

```
Future  0.5
Past   -0.5
```

```
df_tense$gramm <- as.factor(df_tense$gramm)
levels(df_tense$gramm)
```

```
[1] "gramm"   "ungramm"
```

```
contrasts(df_tense$gramm) <- c(-0.5, +0.5)
contrasts(df_tense$gramm)
```

```
        [,1]
gramm   -0.5
ungramm  0.5
```

Now that we've set our contrasts (if you have continuous predictors, you would centre and potentially standardise them instead), we can fit our model. We use the `glm()` function to fit a genearlised linear model, and use the argument `family = "binomial"` to indicate our data are binomial.

```
fit_tense_ri <-
  glm(ri ~ verb_t*gramm,
    data = df_tense,
    family = "binomial")
```

What do our coefficients look like?

```
tidy(fit_tense_ri) %>%
  mutate(p.value = as.numeric(p.value)) |>
  mutate(p.value = round(p.value*4,10)
         ) |>
  knitr::kable() |>
  kableExtra::kable_styling()
```

Let's first consider the estimates. The intercept is negative, meaning it is below 0. Verb tense is positive, meaning that there are more regressions in for the future compared to the past, holding grammaticality constant. Grammaticality is positive, meaning that there were more regressions in for the ungrammatical than grammatical conditions. But what does zero mean here? Logistic regression gives the estimates in log-odds. This means that a value of 0 means

term	estimate	std.error	statistic	p.value
(Intercept)	-1.2472175	0.0392027	-31.8146220	0.0000000
verb_t1	0.0209637	0.0784053	0.2673755	3.1567201
gramm1	0.3668992	0.0784053	4.6795205	0.0000115
verb_t1:gramm1	-0.1197221	0.1568106	-0.7634823	1.7807033

there is an equal probability of a regression in or out for both conditions (as in (**tab-odds?**)), i.e., the slope is flat (or not significantly different from 0). How can we convert our log-odds estimates to something more interpretable, like probabilities? Recall the equation in 6.3, which would require a lot of typing. Luckily, we can just use the `plogis()` function, which takes a log-odds value and spits out the corresponding probability. We can also just use the `exp()` function to get the odds ratio from the log-odds.

```
plogis(-1.23) # intercept prob
```

```
[1] 0.2261814
```

```
plogis(0.0277) # tense prob
```

```
[1] 0.5069246
```

```
exp(-1.23) # intercept odds
```

```
[1] 0.2922926
```

```
exp(0.0277) # tense odds
```

```
[1] 1.028087
```

This is great, but a bit tedious. We can also just feed a tibble column through the `plogis()` and `exp()` functions to print a table with the relevant probabilities and odds.

```
tidy(fit_tense_ri) %>%
  mutate(p.value = round(p.value*4,10),
         prob = plogis(estimate),
         odds = exp(estimate)
```

term	estimate	std.error	statistic	p.value	prob	odds
(Intercept)	-1.2472	0.0392	-31.8146	0.0000	0.2232	0.2873
verb_t1	0.0210	0.0784	0.2674	3.1567	0.5052	1.0212
gramm1	0.3669	0.0784	4.6795	0.0000	0.5907	1.4433
verb_t1:gramm1	-0.1197	0.1568	-0.7635	1.7807	0.4701	0.8872

```
        ) |>
  mutate_if(is.numeric, round, 4) |>
  knitr::kable() |>
  kableExtra::kable_styling()
```

We see that the odds of the future tense have a regression in versus the past tense is ~1, with the corresponding probability of 0.51. Unsuprisingly, we see this p-value indicates this effect was not significant ($p > .05$), and the z-value (note: not t-value!) is also low.

> **i** *z*-values
>
> z-values correspond to the estimate divided by the standard error. It's interpretation is similar to that of the t-value: a z-value of ~2 or higher will likely have a p-value below 0.05.

The interaction term is negative, what does this mean? We can interpret this as indicating that the effect of congruence is different in either level of tense. These effects are often more easily interpreted with a visualisation, e.g., using the `plot_model()` function from the `sjPlot` package. This effect is not significant, however.

```
sjPlot::plot_model(fit_tense_ri,
                   type = "eff",
                   terms = c("gramm", "verb_t")) +
  geom_line(position = position_dodge(0.1))
```

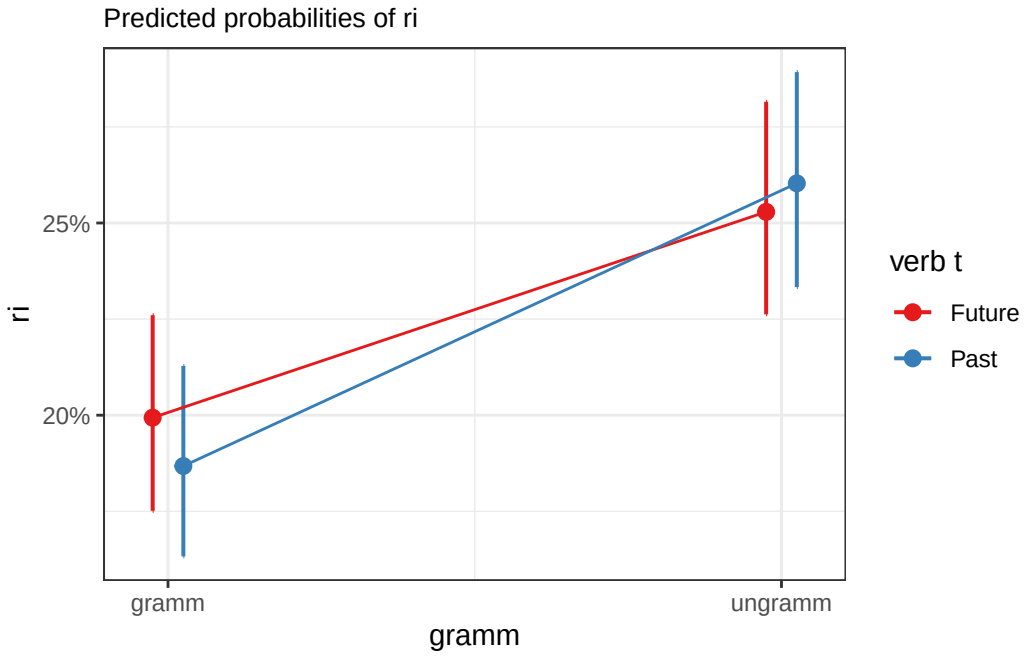


Figure 6.2: Interaction plot of

We can also use the `predict()` function to extract the predicted values for each condition. We could just simply print the predicted values (`predict(fit_tense_ri)`), append the predicted values to the data frame

```
# make sure dataset is the same length as the model data
df_tense_v <-
  df_tense |>
  filter(roi == "4") |>
  drop_na(ri)

# append model estimates
augment(fit_tense_ri, data = df_tense_v) |>
  distinct(verb_t, gramm, .keep_all = T) |>
  arrange(verb_t) |>
  select(verb_t, gramm, .fitted)
```

```
# A tibble: 4 x 3
  verb_t gramm   .fitted
  <fct>  <fct>     <dbl>
1 Future gramm   -1.3903
2 Future ungramm -1.0832
```

```
3 Past   gramm   -1.4711
4 Past   ungramm -1.0443
```

Or we could create a list of the unique conditions.

```
df_sim <-
    tibble(
    verb_t = rep(c('Past', 'Future'), each = 2),
    gramm = rep(c('0', '1'), times = 2))

# alternatively, just extract the relevant factor levels from your datafram
df_sim <-
  df_tense |>
  arrange(verb_t) |>
  distinct(verb_t, gramm)

# and add predicted values
df_sim$fit <- num(predict(fit_tense_ri, df_sim), digits = 5)

df_sim
```

```
# A tibble: 4 x 3
  verb_t gramm         fit
  <fct>  <fct>   <num:.5!>
1 Future gramm    -1.39025
2 Future ungramm  -1.08322
3 Past   gramm    -1.47108
4 Past   ungramm  -1.04432
```

And now if we look at the predicted log-odds values for the future and past tenses:

```
df_sim |>
  summarise(
    mean_tense = mean(fit),
    .by = verb_t)
```

```
# A tibble: 2 x 2
  verb_t mean_tense
  <fct>   <num:.5!>
1 Future   -1.23674
2 Past     -1.25770
```

What is the difference between these two numbers (in our model summary)?

```
df_sim |>
  summarise(
    mean_tense = mean(fit),
    .by = gramm)
```

```
# A tibble: 2 x 2
  gramm   mean_tense
  <fct>     <num:.5!>
1 gramm      -1.43067
2 ungramm    -1.06377
```

What is the difference between these two numbers (in our model summary)?

So, our slopes for `verb_t` and `gramm` correspond to the predicted difference between the two levels of each factor.

6.3 Interpreting our coefficients

What do our coefficient estimates reflect, though? Let's remind ourselves of the rate of regressions in at the verb region:

```
df_tense |>
  filter(roi == "4") |>
  drop_na(ri) |>
  summary()
```

```
     sj                  item          adv_type            adv_t
 Length:3795        Min.   :  1.00   Length:3795        Length:3795
 Class :character   1st Qu.: 25.00   Class :character   Class :character
 Mode  :character   Median : 52.00   Mode  :character   Mode  :character
                    Mean   : 50.91
                    3rd Qu.: 78.00
                    Max.   :101.00
   verb_t         gramm           roi       label                fp
 Future:1897   gramm  :1901   Min.   :4   Length:3795        Min.   :  81.0
 Past  :1898   ungramm:1894   1st Qu.:4   Class :character   1st Qu.: 266.0
                              Median :4   Mode  :character   Median : 371.0
                              Mean   :4                      Mean   : 440.5
```

```
                           3rd Qu.:4                      3rd Qu.: 535.0
                           Max.   :4                      Max.   :2833.0
      gp             tt               ri                ro
 Min.   :  87   Min.   :  90.0   Min.   :0.0000   Min.   :0.00000
 1st Qu.: 285   1st Qu.: 326.5   1st Qu.:0.0000   1st Qu.:0.00000
 Median : 405   Median : 493.0   Median :0.0000   Median :0.00000
 Mean   : 510   Mean   : 607.4   Mean   :0.2248   Mean   :0.08169
 3rd Qu.: 606   3rd Qu.: 747.0   3rd Qu.:0.0000   3rd Qu.:0.00000
 Max.   :3877   Max.   :3936.0   Max.   :1.0000   Max.   :1.00000
```

```r
ptab_gramm <-
  df_tense |>
  filter(roi == "4") |>
  drop_na(ri) |>
  select(gramm, ri) |>
  table() |>
  prop.table()

ptab_tense <-
  df_tense |>
  filter(roi == "4") |>
  drop_na(ri) |>
  select(verb_t, ri) |>
  table() |>
  prop.table()

df_tense |>
  filter(roi == "4") |>
  drop_na(ri) |>
  tabyl(gramm, ri, verb_t) |>
  adorn_percentages() |>
  adorn_totals()
```

```
$Future
   gramm         0         1
   gramm 0.8006329 0.1993671
 ungramm 0.7471022 0.2528978
   Total 1.5477351 0.4522649

$Past
   gramm         0         1
   gramm 0.8132214 0.1867786
```

```
ungramm 0.7396825 0.2603175
  Total 1.5529039 0.4470961
```

We want to measure how much more likely a regression in (y = 1) is for ungrammatical conditions (x = 1) than in grammatical conditions (x = 0). Si we want to calculate the odds of a regression in for each case, and take their ratio:

```
# odds(y = 1 | x = 0)
odds_ri1_gramm0 <-
  ptab_gramm[1,2] / ptab_gramm[1,1] # in gramm conditions: ri 0/1
odds_ri1_gramm1 <-
  ptab_gramm[2,2] / ptab_gramm[2,1] # in ungramm condiitons: ri 0/1

## odds ratio
odds_ri1_gramm1 / odds_ri1_gramm0
```

```
[1] 1.442756
```

```
## log odds
log(odds_ri1_gramm1) - log(odds_ri1_gramm0)
```

```
[1] 0.3665552
```

```
# or
log(odds_ri1_gramm1 / odds_ri1_gramm0)
```

```
[1] 0.3665552
```

```
## probability
plogis(log(odds_ri1_gramm1 / odds_ri1_gramm0))
```

```
[1] 0.5906263
```

So the odds of a regression into the verb region is 1.4 times more likely in ungrammatical versus grammatical conditions.

```
intercept <- tidy(fit_tense_ri)$estimate[1]
tense <- tidy(fit_tense_ri)$estimate[2]
gramm <- tidy(fit_tense_ri)$estimate[3]
interact <- tidy(fit_tense_ri)$estimate[4]
```

What are the log odds for the past (tense = -0.5) grammatical (`gramm` = -0.5)?

```
plogis(intercept)
```

```
[1] 0.2231822
```

```
plogis(tense)
```

```
[1] 0.5052407
```

```
plogis(gramm)
```

```
[1] 0.5907095
```

```
plogis(interact)
```

```
[1] 0.4701052
```

```
tidy(fit_tense_ri) |>
  mutate(prob = plogis(estimate))
```

```
# A tibble: 4 x 6
  term            estimate std.error statistic     p.value    prob
  <chr>              <dbl>     <dbl>     <dbl>       <dbl>   <dbl>
1 (Intercept)    -1.2472    0.039203 -31.815   4.0637e-222 0.22318
2 verb_t1         0.020964  0.078405   0.26738 7.8918e-  1 0.50524
3 gramm1          0.36690   0.078405   4.6795  2.8755e-  6 0.59071
4 verb_t1:gramm1 -0.11972   0.15681   -0.76348 4.4518e-  1 0.47011
```

```
plogis(intercept + tense*-.5 + gramm*-.5)
```

```
[1] 0.1913675
```

And past ungrammatical (gramm = +0.5)?

```
plogis(intercept + tense*-.5 + gramm*.5)
```

```
[1] 0.2545957
```

And for the future conditions?

```
plogis(intercept + tense*.5 + gramm*-.5)
```

```
[1] 0.1946325
```

And past ungrammatical (gramm = +0.5)?

```
plogis(intercept + tense*.5 + gramm*.5)
```

```
[1] 0.2585946
```

```
plogis(-1.22521)
```

```
[1] 0.2270209
```

```
plogis(-1.22521)
```

```
[1] 0.2270209
```

$$p = \frac{odds}{1 + odds} \tag{6.4}$$

$$odds = \frac{p}{1 - p} \tag{6.5}$$

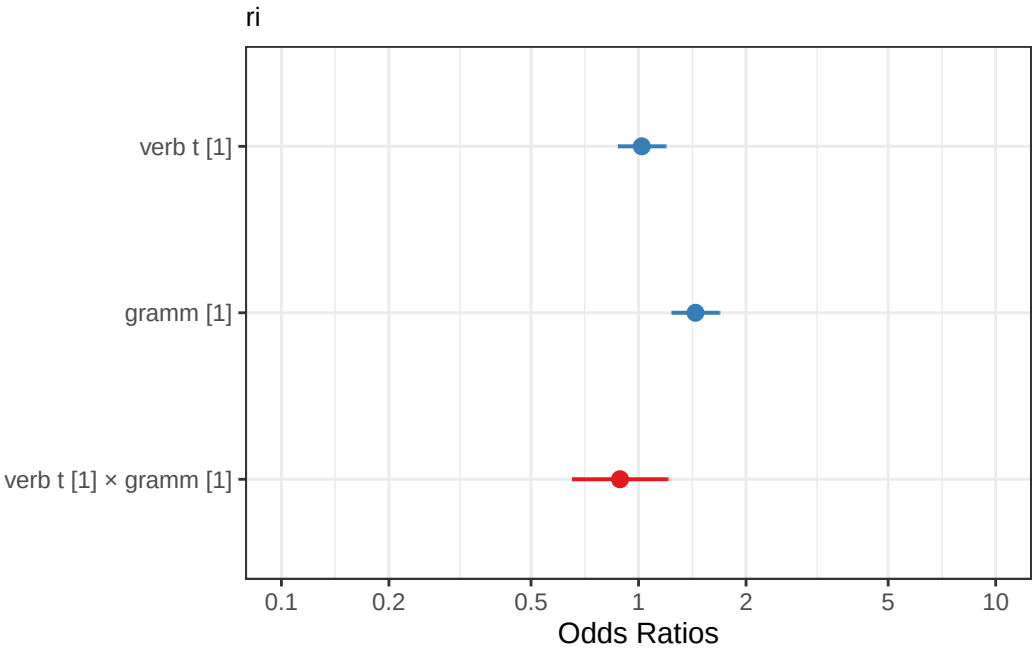
$$log\ odds = exp(odds) \tag{6.6}$$

6.4 Visualising model predictions

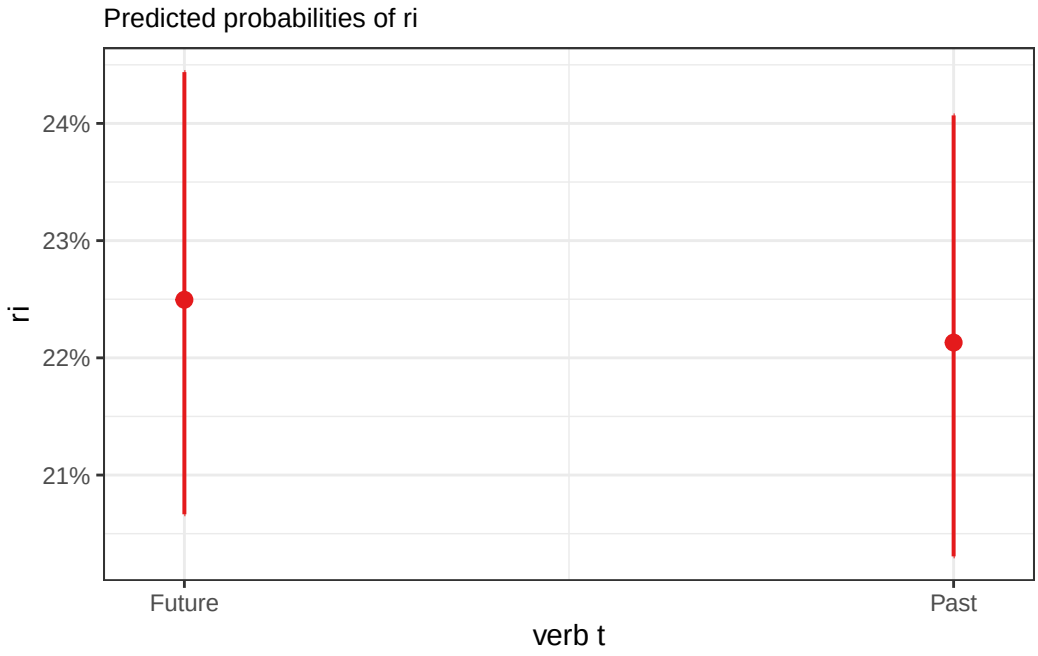
Something we haven't really covered is how to visualise our model predictions. So far we've only visualised the raw data, but when interpreting model results it helps to see the predictions. This is especially true for logistic regression, because our estimates are given in log odds, which are not very intuitive.

We can use the `sjPlot` package, which is very handy:

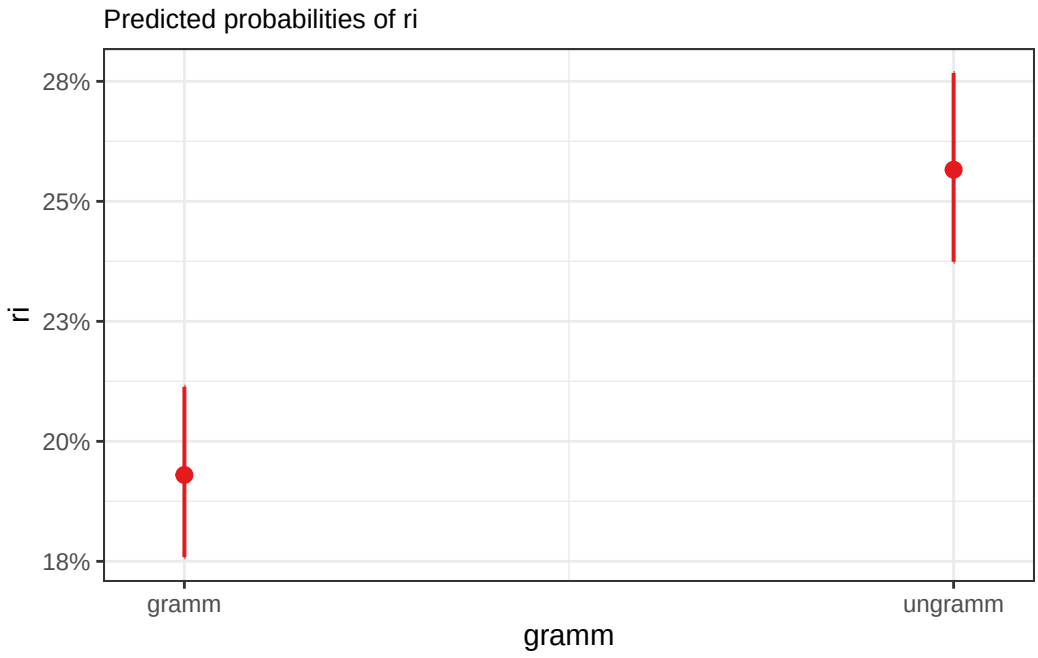
```
library(sjPlot)

plot_model(fit_tense_ri)
```



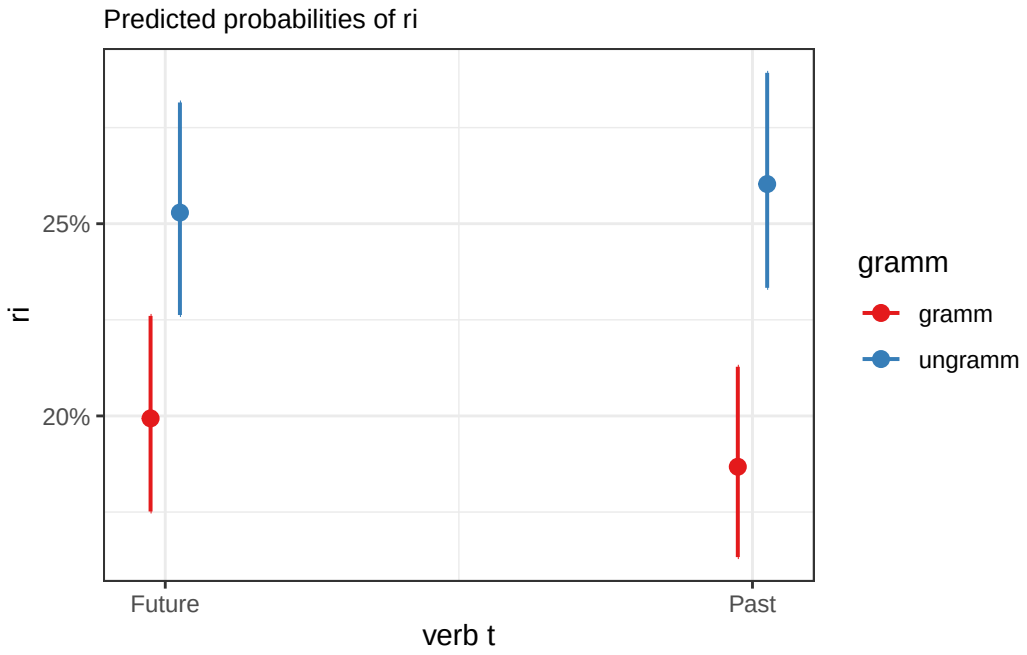

```
plot_model(fit_tense_ri, type = "eff",
           terms = "verb_t")
```




```
plot_model(fit_tense_ri, type = "eff",
           terms = "gramm")
```



```
plot_model(fit_tense_ri, type = "int")
```



Or we can use the `ggeffects` package to extract summaries of effects, and then feed them into `ggplot2`.

```
library(ggeffects)
```

```
ggeffect(fit_tense_ri)
```

```
$verb_t
# Predicted probabilities of ri

verb_t | Predicted |       95% CI
--------------------------------
Future |      0.22 | [0.21, 0.24]
Past   |      0.22 | [0.20, 0.24]

$gramm
# Predicted probabilities of ri

gramm   | Predicted |       95% CI
```

```
---------------------------------
gramm   |      0.19 | [0.18, 0.21]
ungramm |      0.26 | [0.24, 0.28]

attr(,"class")
[1] "ggalleffects" "list"
attr(,"model.name")
[1] "fit_tense_ri"
```

```
ggeffect(fit_tense_ri,
         terms = c("gramm", "verb_t"))
```

```
# Predicted probabilities of ri

# verb_t = Future

gramm   | Predicted |       95% CI
---------------------------------
gramm   |      0.20 | [0.18, 0.23]
ungramm |      0.25 | [0.23, 0.28]

# verb_t = Past

gramm   | Predicted |       95% CI
---------------------------------
gramm   |      0.19 | [0.16, 0.21]
ungramm |      0.26 | [0.23, 0.29]
```

6.5 Reporting

Sonderegger (2023) (Section 6.9) says the following:

> Reporting a logistic regression model in a write-up is generally similar to reporting a linear regression model…Reporting a logistic regression model in a write-up is generally similar to reporting a linear regression model: the guidelines and rationale in section 4.6 for reporting individual coefficients and the whole model hold, with some adjustments.

We can produce such a table using the `papaja` package, as in Table 6.2.

Table 6.2: **?(caption)**

(a) Model summary for regressions in at the verb region. Estimates are given in log odds.

Predictor	*b*	95% CI	*z*	*p*
Intercept	-1.25	[-1.32, -1.17]	-31.81	< .001
Verb t1	0.02	[-0.13, 0.17]	0.27	.789
Gramm1	0.37	[0.21, 0.52]	4.68	< .001
Verb t1 × Gramm1	-0.12	[-0.43, 0.19]	-0.76	.445

Table 6.3: **?(caption)**

(a)

(b)

term	estimate	std.error	prob	statistic	p.value
(Intercept)	-1.25	0.04	0.22	-31.81	0.00
verb_t1	0.02	0.08	0.51	0.27	0.79
gramm1	0.37	0.08	0.59	4.68	0.00
verb_t1:gramm1	-0.12	0.16	0.47	-0.76	0.45

```
library(papaja)

fit_tense_ri |>
  apa_print() |>
  apa_table(caption = "Model summary for regressions in at the verb region. Estimates are
```

Or by extracting the model summary with `tidy()`, and even adding our probabilities, as in Table 6.3.

```
tidy(fit_tense_ri) |>
  mutate(prob = plogis(estimate)) |>
  relocate(prob, .after = std.error) |>
  apa_table()
```

6.6 Summary

- we saw that the equation for a straight line boils down to its intercept and slope

term	description/other terms

- we fit our first linear model with a categorical predictor

Important terms

Important terms

Term	Definition	Equation/Code
NA	NA	NA

Learning Objectives

Today we learned...

- how to model binomial data with logistic regression
- how to interpret log-odds and odds ratio

Task

6.6.1 Regressions out

Using the same dataset, run a logistic model exploring regressions in (`ri`) at the adverb region (`roi = "2"`). Before you run the model, do you have any predictions? Try plotting the regressions in for this region first, and generate some summary tables to get an idea of the distributions of regressions in across conditions.

6.6.2 Dutch verb regularity

Load in the `regularity` data from the `languageR` package.

```
df_reg <-
  regularity |>
  clean_names()
```

Regular and irregular Dutch verbs and selected lexical and distributional properties.

Our relevant variables will be:

- `written_frequency`: a numeric vector of logarithmically transformed frequencies in written Dutch (as available in the CELEX lexical database).
- `regularity`: a factor with levels irregular (1) and regular (0).
- `verb`: a factor with the verbs as levels.

1. Fit a logistic regression model to the data which predicts verb regularity by written frequency. Consider: What type of predictor variable do you have, and what steps should you take before fitting your model?
2. Print the model coefficients, e.g., using `tidy()`.
3. Interpret the coefficients, either in log-odds or probabilities. Report your findings.

Literaturverzeichnis

Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics using R*.

Baayen, R. H., Davidson, D. J., & Bates, D. M. (2008). Mixed-effects modeling with crossed random effects for subjects and items. *Journal of Memory and Language*, *59*(4), 390–412. https://doi.org/10.1016/j.jml.2007.12.005

Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing linguistic data: A practical introduction to statistics*. https://CRAN.R-project.org/package=languageR

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language*, *68*(3), 255–278. https://doi.org/10.1016/j.jml.2012.11.001

Bates, D., Kliegl, R., Vasishth, S., & Baayen, H. (2015). Parsimonious Mixed Models. *arXiv Preprint*, 1–27. https://doi.org/10.48550/arXiv.1506.04967

Biondo, N., Soilemezidi, M., & Mancini, S. (2022). Yesterday is history, tomorrow is a mystery: An eye-tracking investigation of the processing of past and future time reference during sentence reading. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *48*(7), 1001–1018. https://doi.org/10.1037/xlm0001053

Jaeger, T. F. (2008). Categorical data analysis: Away from ANOVAs (transformation or not) and towards logit mixed models. *Journal of Memory and Language*, *59*(4), 434–446. https://doi.org/10.1016/j.jml.2007.11.007

Lüdecke, D., Ben-Shachar, M. S., Patil, I., Waggoner, P., & Makowski, D. (2021). performance: An R package for assessment, comparison and testing of statistical models. *Journal of Open Source Software*, *6*(60), 3139. https://doi.org/10.21105/joss.03139

Matuschek, H., Kliegl, R., Vasishth, S., Baayen, H., & Bates, D. (2017). Balancing Type I error and power in linear mixed models. *Journal of Memory and Language*, *94*, 305–315. https://doi.org/10.1016/j.jml.2017.01.001

Sonderegger, M. (n.d.). *Regression Modeling for Linguistic Data*.

Sonderegger, M. (2023). *Regression Modeling for Linguistic Data*.

Vasishth, S. (2022). *Some right ways to analyze (psycho)linguistic data* [Preprint]. PsyArXiv. https://doi.org/10.31234/osf.io/y54va

Vasishth, S., & Nicenboim, B. (2016). Statistical methods for linguistic research: Foundational Ideas—Part I. *Language and Linguistics Compass*, *10*(11), 591–613. https://doi.org/10.1111/lnc3.12207

Winter, B. (2013). *Linear models and linear mixed effects models in R: Tutorial 1*.

Winter, B. (2014). *A very basic tutorial for performing linear mixed effects analyses (Tutorial 2)*.

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

Part V

Report

7 Report 1

Linear and logistic regression

The goal of this report is to review and consolidate what we learned together in the first block of the course. You are not required to do anything that we have not already seen.

For students enrolled in this course in the Winter Semester 2023/24: The report is due **January 11, 2024 at 11:59pm**, but I encourage you to submit before the holidays if you have the capacity to do so. Please submit your Quarto script, as well as a rendered copy (preferably HTML, but PDF is also fine) to Moodle (under 'Reports').

7.1 Dataset

For this report you will continue using the data from Biondo et al. (2022), an eye-tracking reading study on adverb-tense congruence effects on reading time measures. Participants' eye movements were recorded as they read Spanish sentences where temporal adverbs and verb tense were either congruent or incongruent. For both sentence regions, the time reference was either past (e.g., yesterda, bought) or future (e.g., tomorrow, will buy). Example stimuli from this experiment are given in Table 7.1. You will be fitting models to different eye-tracking reading measures from this experiment, with the predictors *verb tense* and *grammaticality*.

Table 7.1: Example stimuli

sentence	adverb	verb	gramm
A la salida del trabajo, **ayer** las chicas **compraron** pan en la tienda. *After leaving work* **yesterday** *the girls* **bought** *bread at the shop*	past	past	gramm
A la salida del trabajo, **ayer** las chicas ***comprarán** pan en la tienda. *After leaving work* **yesterday** *the girls* ***will buy** *bread at the shop*	past	future	ungramm
A la salida del trabajo, **mañana** las chicas **comprarán** pan en la tienda. *After leaving work* **tomorrow** *the girls* **will buy** *bread at the shop*	future	future	gramm
A la salida del trabajo, **mañana** las chicas ***compraron** pan en la tienda. *After leaving work* **tomorrow** *the girls* ***bought** *bread at the shop*	future	past	ungramm

7.2 Set-up

Make sure you begin with a *clear* working environment. To achieve this, you can go to `Session > Restart R`. Your Environment should have no objects in it, and you should not have any packages loaded.

7.2.1 Packages

Load the packages below. Give a short description of why we load in each package, i.e., what these packages help us do (1-2 sentences each). Tip: remember you can type `?tidyverse` into the Console to get an overview of a package or function.

```
pacman::p_load(
  tidyverse,
  janitor,
  here,
  broom
)
```

1. `tidyverse`:
2. `janitor`:
3. `here`:

7.2.2 Data

Run the code below. Give a short description of what each line of code does (you can skip the `locale` line). Tip: `roi == 2` corresponds to the *temporal adverb* condiiton.

```
df_tense <-
  read_csv(here("data", "Biondo.Soilemezidi.Mancini_dataset_ET.csv"),
           locale = locale(encoding = "Latin1") ## for special characters in Spanish
           ) |>
  clean_names() |>
  mutate(gramm = ifelse(gramm == "0", "ungramm", "gramm"))  |>
  filter(roi == 2,
         adv_type == "Deic") |>
  mutate(length = nchar(label))
```

You can write your answer like this, for example:

1. `df_tense <-` :

2. `read_csv`:
3. `clean_names`:
4. `mutate`:
5. `filter`:

7.3 How to report your models

You will be running linear and logisitc regression models. Our variables of interest will be:

variable	description	type	class
`fp`	first-pass reading time (summation of fixations from when a reader first fixates on a region to when they first leave that region)	dependent	continuous
`tt`	total reading time (summation of all fixations within a region during a trial)	dependent	continuous
`ri`	regressions in (whether there was at least one regression into a region)	dependent	binomial
`ro`	regressions out (whether there was at least one regression out of a region)	dependent	binomial
`verb_t`	verb tense: past or future	independent	categorical
`gramm`	grammaticality: grammatical or ungrammatical	independent	categorical
`length`	region length in letters	independent	continuous

7.3.1 Example model report

Imagine we fit a linear model, called `fit_verb_tt`, to log-tranformed first-pass reading times at the verb region. Our fixed effects (i.e., predictors) are verb tense, grammaticality, and their interaction. Below I report the findings of the model, which is what you should aim to do with the models you run.

> The model summary is given in **?@tbl-fit_verb_tt**, with back-transformed model predictions visualised in Figure 7.1. A main effect of grammaticality was found in total reading times at the verb region, with ungrammatical conditions eliciting longer total reading times than grammatical conditions (Est = -0.06, t = -2.4, p < 0.05). A main effect of tense was also found, with the future condition eliciting longer total reading times than the past condition (Est = 0.07, t = 2.48, p < .05). An interaction of tense and grammaticality was not significant (Est = 0.04, t = 1).

```
library(papaja)

fit_verb_tt |>
```

Table 7.3: Model summary for (log-transformed) total reading times at the verb region.

Predictor	*b*	95% CI	*t*	*df*	*p*
Intercept	6.22	[6.19, 6.26]	332.72	3791	$< .001$
Verb tPast	-0.06	[-0.11, -0.01]	-2.38	3791	.017
Grammungramm	0.07	[0.01, 0.12]	2.47	3791	.013
Verb tPast × Grammungramm	0.04	[-0.04, 0.11]	1.01	3791	.312

```
  apa_print() |>
  apa_table(label = "tbl-fit_verb_tt",
            caption = "Model summary for (log-transformed) total reading times at the verb
```

```
library(sjPlot)
plot_model(fit_verb_tt, type = "int") +
  geom_line(position = position_dodge(0.1)) +
  labs(title = "Predicted total reading times at the verb region",
       x = "Verb tense",
       y = "Reading time (ms)") +
  theme_bw()
```

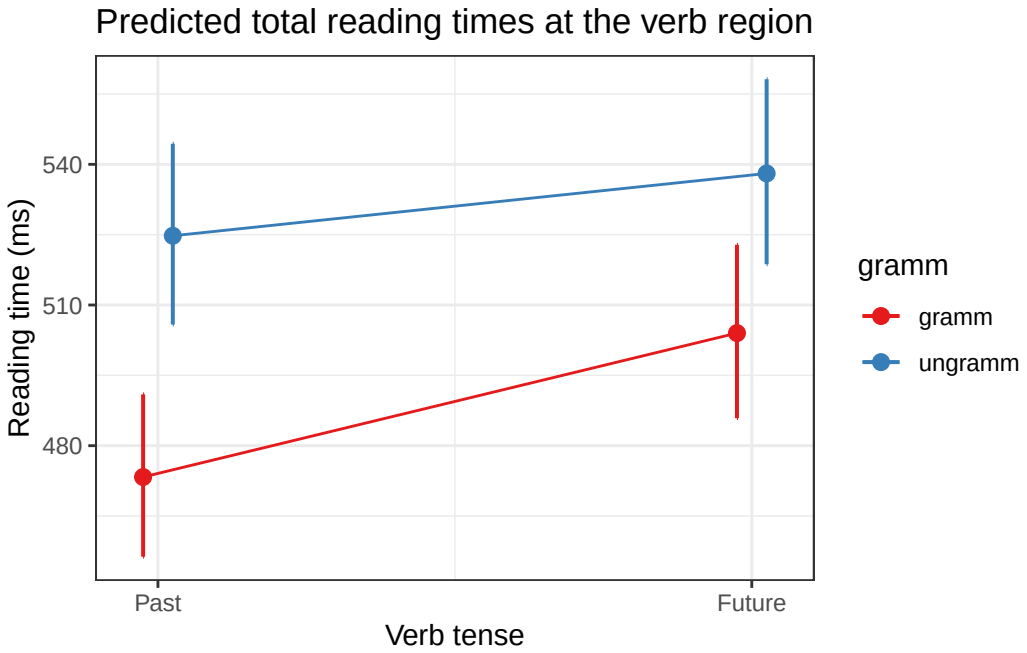


Figure 7.1: Back-transformed model predictions for total reading time at the verb region (with 95% confidence intervals).

7.4 Variable prep

We need to prepare our predictors: centering continuous variables and sum contrast coding categorical variables.

7.4.1 Centring continuous variables

Create a new variable `length_c` which contains the centred values of `length` (just centre, you don't need to standardise). Centre using the *median* rather than the mean (hint: there is a function `median()`).

7.4.2 Contrast coding

Set sum contrast coding for `tense` and `gramm`. You might need to first use the `as.factor()` function to save the variables as factors. Your contrasts should look like this:

```
contrasts(df_tense$gramm)
```

```
        [,1]
gramm   -0.5
ungramm  0.5
```

```
contrasts(df_tense$verb_t)
```

```
       [,1]
Future  0.5
Past   -0.5
```

7.5 Linear regression

We run linear regression when we have a continuous dependent variable. You will be fitting a. model to first-pass reading time.

7.5.1 Fitting our model

Fit a model of *log-transformed* first-pass reading times with verb tense, grammaticality, and their interaction as fixed effects (hint: you might want to use `*` in your model).

7.5.2 Assessing assumptions

Visually assess the model assumptions of normality and homoscedasticity and write 1-2 sentences about each assumption, referring to the figures you produced.

7.5.3 Extracting predictions

1. Create objects `intercept`, `b1` (`verb_t`), and `b2` (`gramm`) that contain the corresponding model coefficient estimate for each term (hint: each object should contain a single value, which corresponds to the *estimate* for this term in your model summary output).
2. Generate the fitted values for each of our four conditions. We covered a number of ways to do this in class:
    (a) Using our model formula (7.1) and the sum contrast values for each level of `verb_t` and `gramm` (i.e., +/-0.5, given in Table 7.4), compute the *back-transformed* predicted total reading time for each condition. Recall: b_0 is our intercept, b_1 is our slope for `verb_t` (i.e., the value of the object you just named `verb_t`), and b_2 is our slope (estimate) for `gramm` (i.e., the value of the object you just named `gramm`).

(b) The `ggeffects` package: we used the `ggeffect()` function, but the `ggpredict()` function back-transforms the estimates for us.

$$tt = exp(b_0 + b_1 * verb\_t + b_2 * gramm) \tag{7.1}$$

Table 7.4: Corresponding value of factor levels to be plugged into equation ef{eq-tt}

	-0.5	+0.5
`verb_t`	past	future
`gramm`	gramm	ungramm

7.5.4 Report model

Write a short report of the model findings. Produce a table and plot like in the example above to supplement your report.

7.6 Logistic regression

We run logistic regression when we have a binimial dependent variable. You will be fitting a logistic regression model to regression in.

7.6.1 Fit model

Fit a generalied linear model (logistic regression) to the regression in data, with the same fixed effects as your linear model above (`verb_t`, `gramm`, and their interaction). Remember, you will need a different function (not `lm`), and to add another argument (`family = ...`).

7.6.2 Interpretation

Write a short report of the model findings. Produce a table and plot like in the example above to supplement yor report. Recall that our coefficient estimates are in *log odds*).

Part VI

Day 4: Mixed models I

Part VII

Day 4: Mixed models II

Part VIII

Day 5: TBD