

Equation of a line

WiSe23/24

Daniela Palleschi

2023-10-10

Table of contents

Set-up environment	2
Load in data	2
Simple linear model: $RT \sim \text{frequency}$	2
Mini-EDA	3
<code>lm()</code>	4
Interpreting model output	6
Adding a fixed effect (slope)	9
Fit model (treatment contrasts)	10
Model assumptions	13
Normality	13
Constant variance	14
Visualising model assumptions	14
performance package	17
Exploring the model	19
Exploring the model: residuals	19
Exploring the model	19
Reporting your model	20
Summary	20
Important terms	20
Task	20

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 suppress 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.

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

Load in data

Now that we've loaded our packages, we can take a look at our dataset. Fitting our data to a model typically follows an Exploratory Data Analysis (EDA), which consists of plotting and summarising a data set. We won't get into the EDA process, but there are many great resources on how to go about it in R (e.g., <https://r4ds.hadley.nz/>).

Let's load our dataset using the `read_csv()` function from the **tidyverse** package **readr**. Alternatively, we could use base R to load in the data with `read.csv()`. Some additional functions in the code below:

- `mutate_if(is.character, as.factor)` force character variables to factors
- filter for the verb region from critical items only, remove participant 3, and remove values of first-fixation that are 0

Simple linear model: $RT \sim \text{frequency}$

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 will be:

$$RT \sim \text{frequency}$$

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).

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.

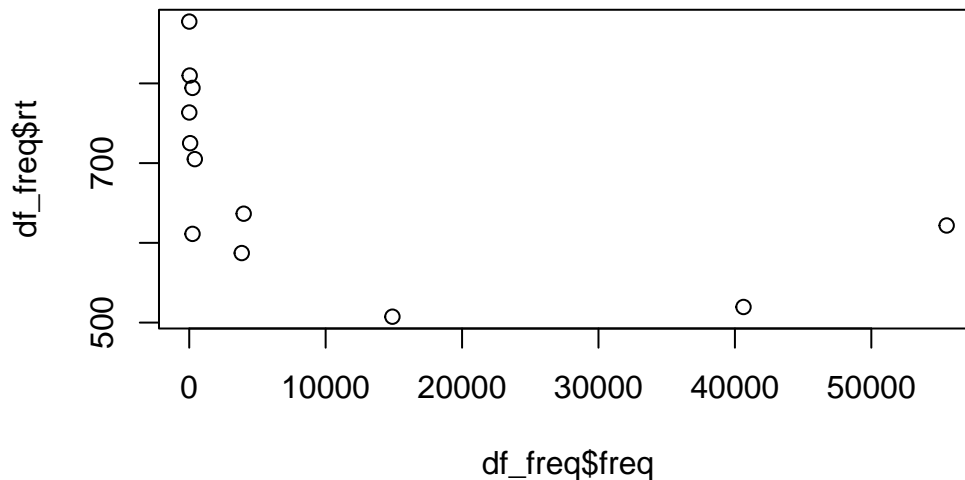
```
# A tibble: 6 x 3
  word    freq    rt
  <chr>   <dbl> <dbl>
1 thing  55522  622.
2 life   40629  520.
3 door   14895  507.
4 angel   3992  637.
5 beer   3850  587.
6 disgrace  409  705
```

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.

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**.



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.

`lm()`

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

$$lm(outcome \sim 1 + predictor, data = df_name)$$

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**.

Running a model

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

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

💡 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.

Model output

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

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.

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

```
# 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

# 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).

Interpreting model output

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

Intercept

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

```
(Intercept)
      679.9167
```

```
[1] 679.9167
```

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

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.

Standard Error

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

$$SE = \frac{\hat{\sigma}}{\sqrt{n}} \quad (1)$$

where σ is the standard deviation, and n is the sample size. As a refresher, the equation for standard deviation (Equation 2) 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 these values come from and what they represent.

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

t-values

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

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

# 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.

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 confident 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.

Plotting $rt \sim 1$

- Figure 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?

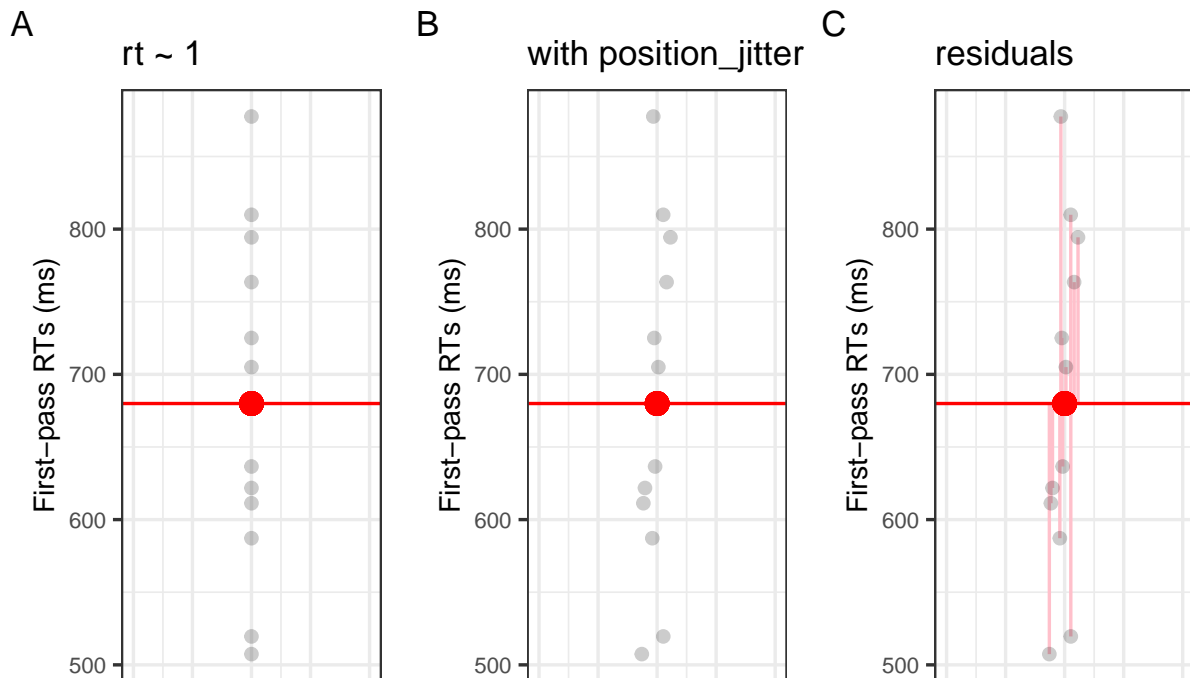
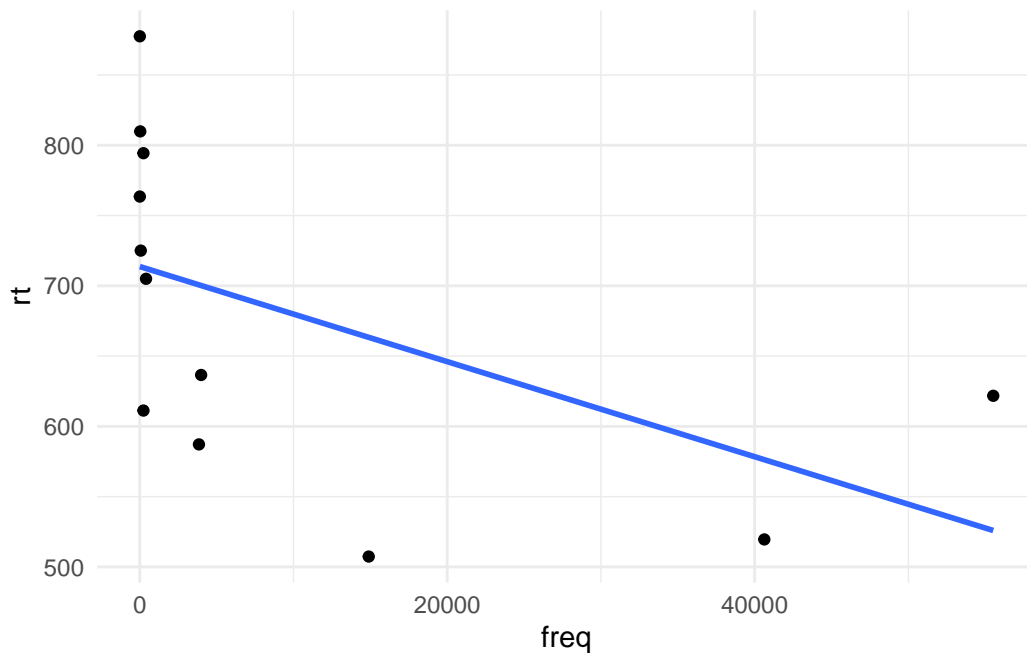


Figure 1: Visualisation of 'rt ~ 1': observed values (black) and mean (intercept; red). Residuals would be the distance from each black dot to the y-value of the read dot

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 x (IV: `freq`). In other words, it tells us the *effect* our IV has on the DV. Let's first plot the data:



Fit model (treatment contrasts)

Model summary

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

Intercept

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

```
(Intercept)
      713.7063
```

```
[1] 679.9167
```

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

Slope

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

```
      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.

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

```
Coefficients:
(Intercept)      year
      396.62      64.58
```

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

```
Coefficients:
(Intercept)      days
      39.66230      0.01769
```

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

```
Coefficients:
(Intercept)      months
  0.396623      0.005382
```

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

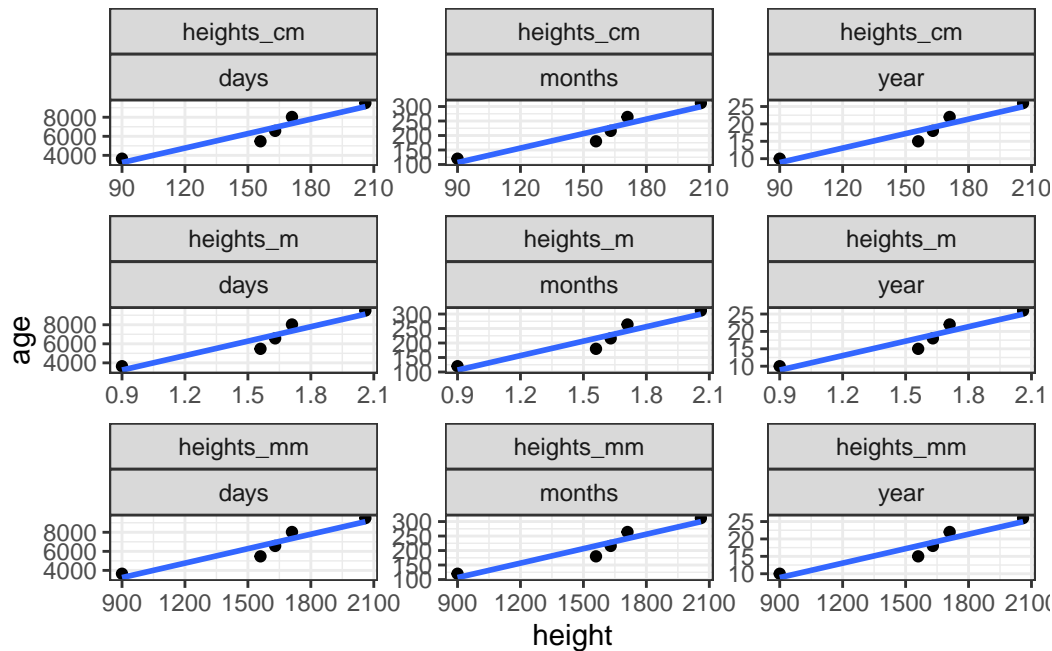
```
Coefficients:
(Intercept)      year
   396.62      64.58
```

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

```
Coefficients:
(Intercept)      year
   39.662      6.458
```

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

```
Coefficients:
(Intercept)      year
  0.39662      0.06458
```



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 relate 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.

Normality

When a model satisfies the normality 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 A) and/or a quantile-quantile (Q-Q) plot (Figure 2 B).

i 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

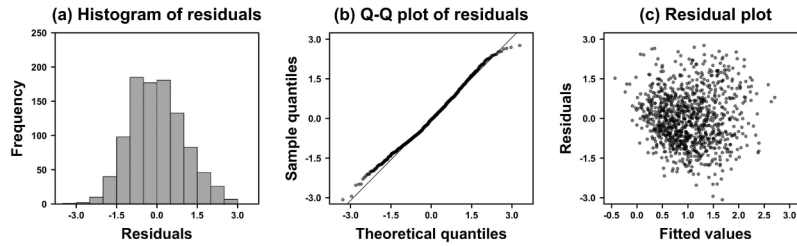


Figure 6.1. (a) Histogram, (b) Q-Q plot, and (c) residual plot of `icon_md1`

Figure 2: Image source: Winter (2019) (all rights reserved)

line, which indicates that the two distributions (the distribution of the residuals and the theoretical normal distribution) are very similar.

Constant variance

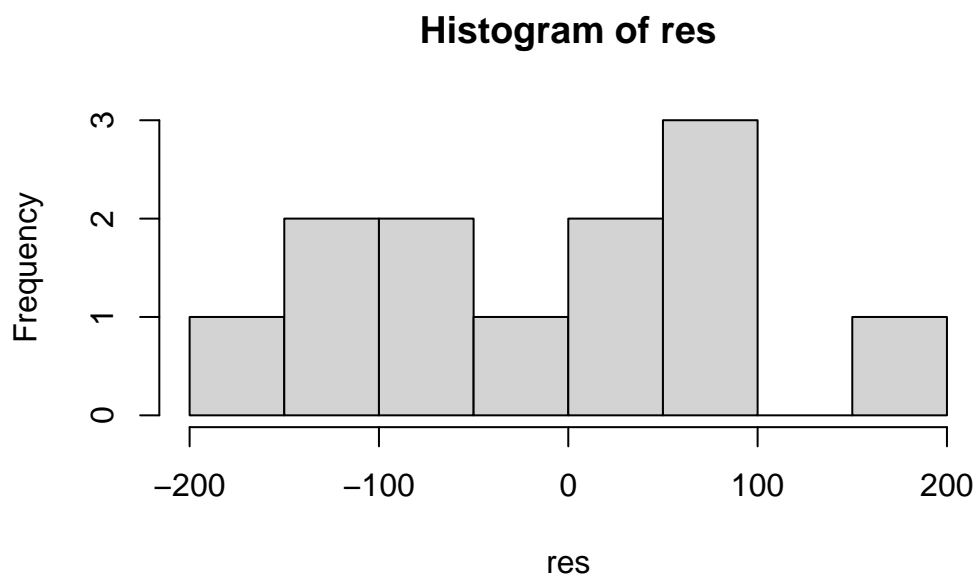
When a model satisfies the constant variance assumption (also called *homoscedasticity*, or the absence of *heteroscedasticity*), the spread of residuals will be equal across the regression line. This is typically visualised using a residual plot, which should look like a blob (Figure 2 C).

Visualising model assumptions

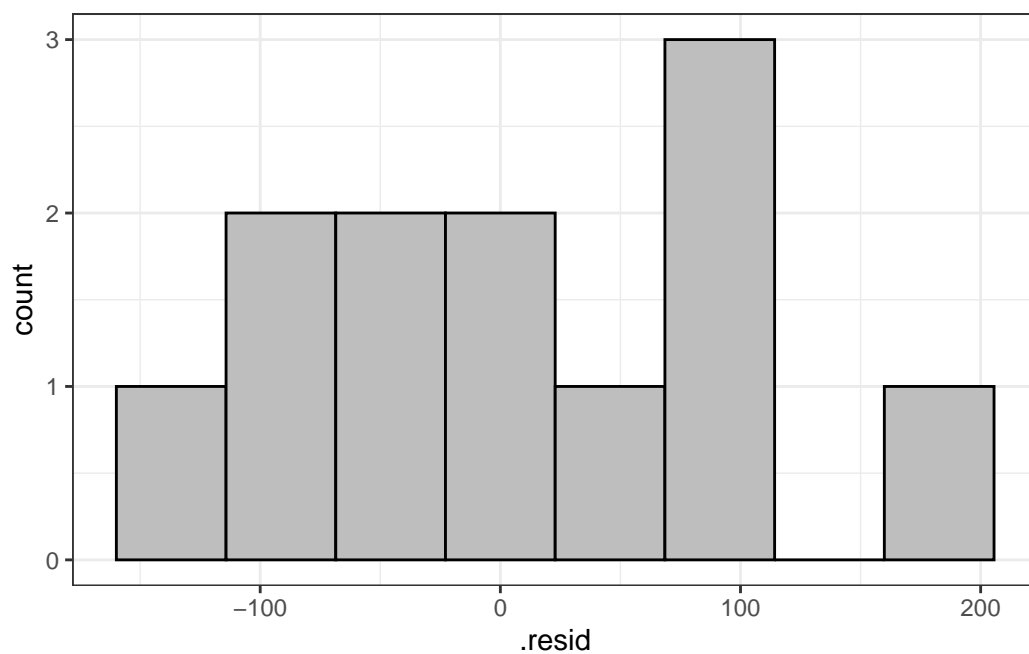
Let's plot our residuals to assess whether our model satisfies the assumptions of normality and constant variance.

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 using the base R function `hist()`.



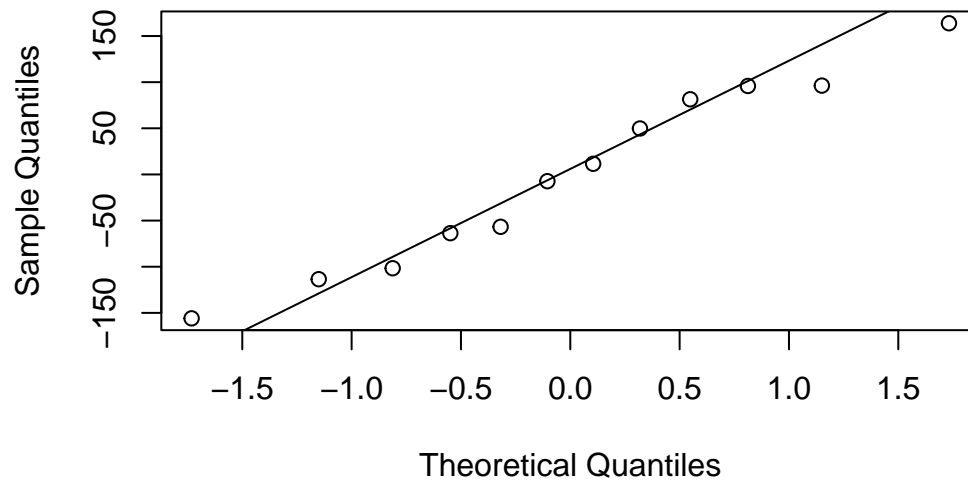
Or, we can use the `augment()` function from `broom` to append model values to our original data frame, and then feed this into `ggplot()` from `ggplot2` (or even feed it into `hist()`).



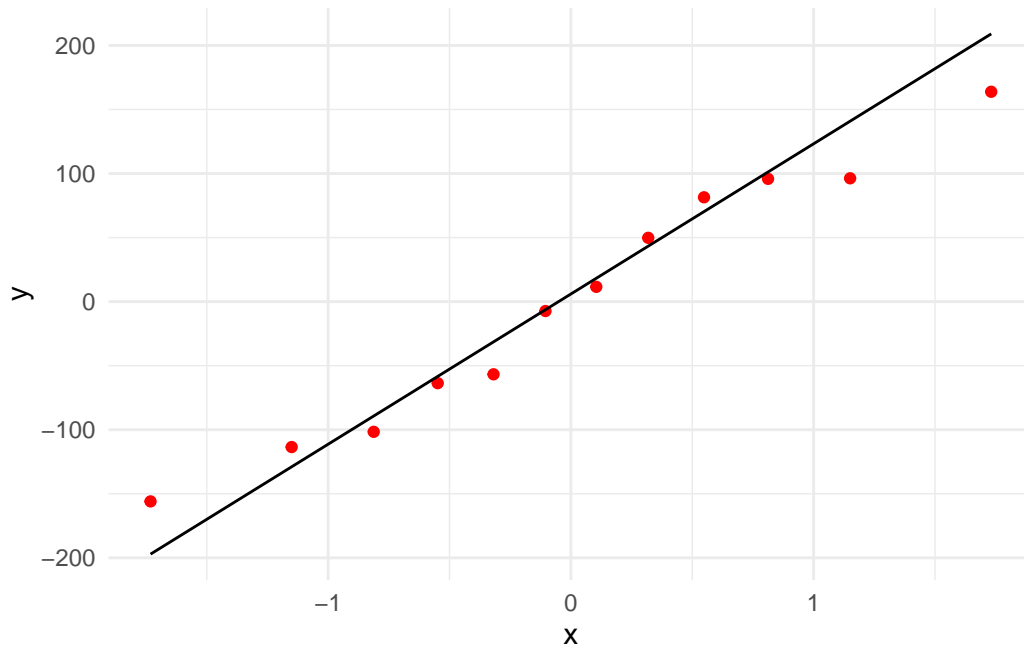
Q-Q plot

Again, we can do it Bodo's way:

Normal Q-Q Plot

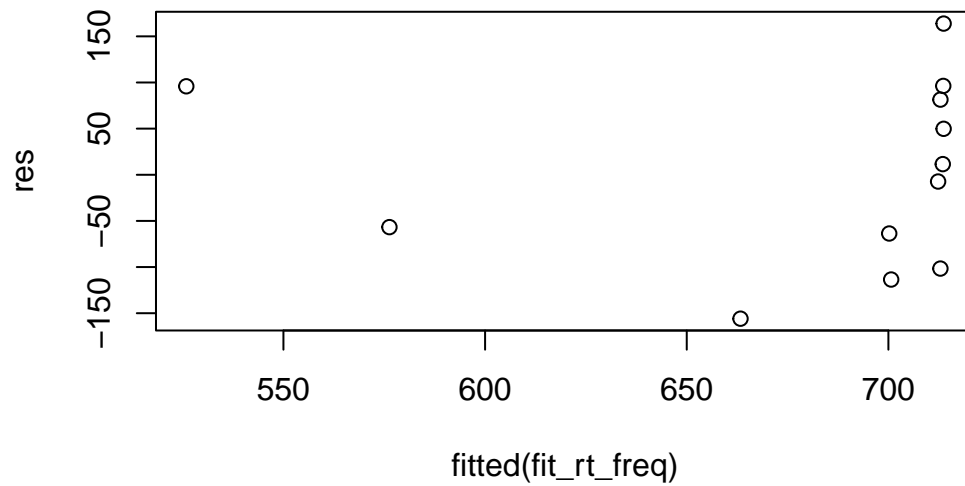


Or using `augment()` and `ggplot()`.

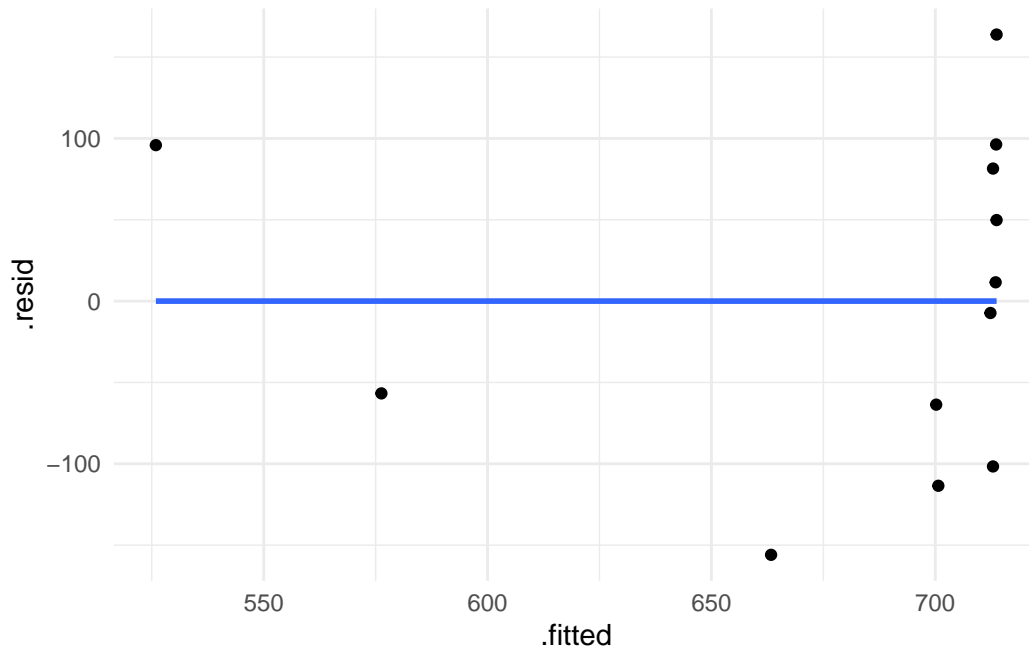


Residual plot

Bodo's way:



Or with ggplot:



performance package

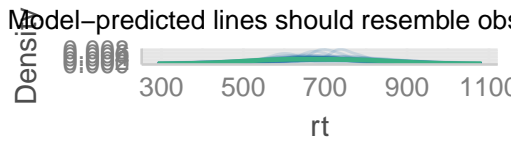
I like to use the `performance` package to visualise model fit (Lüdtke et al., 2021).

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

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

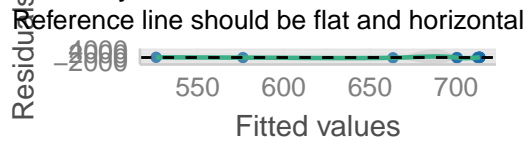
Posterior Predictive Check

Model-predicted lines should resemble observed data



Linearity

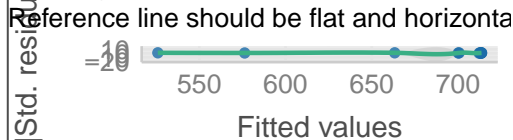
Reference line should be flat and horizontal



— Observed data — Model-predicted

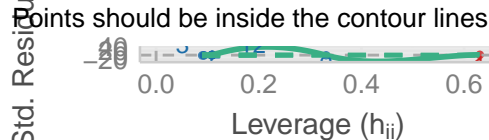
Homogeneity of Variance

Reference line should be flat and horizontal



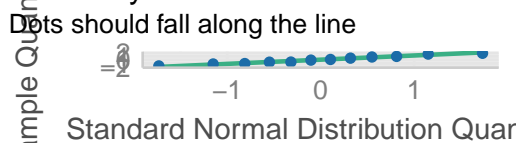
Influential Observations

Points should be inside the contour lines



Normality of Residuals

Points should fall along the line



Standard Normal Distribution Quantiles

Coefficients table with `summary()`

```
> summary(fit_rt_freq)
```

Call:

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

Residuals:

Min	1Q	Median	3Q	Max
-228.99	-109.29	-26.99	58.86	777.71

#<2>

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	309.142	6.259	49.394	<0.0000000000000002 ***
lifetime1	31.701	12.517	2.533	0.0116 *

#<3>

#<4>

#<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

```
[1] 12
```

```
[1] 12
```

Exploring the model: residuals

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

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

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

```
      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?

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

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

```
(Intercept)
696.7949
```

Reporting your model

Section

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	Equation/Code
Simple linear regression	NA	<code>'lm(response ~ predictor, data = data)'</code>

Learning Objectives

Today we learned...

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.

Table A1. Correspondences between some significance tests and linear models

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

Table A2. Correspondences between ANOVAs and linear models

ANOVA	Linear model	Description
<code>aov(y ~ c3)</code>	<code>lm(y ~ c3)</code>	One-way ANOVA with three-level factor
<code>aov(y ~ c2 * c2)</code>	<code>lm(y ~ c2 * c2)</code>	2 x 2 ANOVA (two-way ANOVA)
<code>aov(y ~ c2 * c3)</code>	<code>lm(y ~ c2 * c3)</code>	2 x 3 ANOVA (and so on)
<code>aov(y ~ c2 * covariate)</code>	<code>lm(y ~ c2 * covariate)</code>	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 3: Image source: Winter (2019) (all rights reserved)

Literaturverzeichnis

- 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>
- 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>