

# Simple linear regression

WiSe23/24

Daniela Palleschi

2023-10-10

## Table of contents

```
library(broman)
# function to format p-values
format_pval <- function(pval){
  dplyr::case_when(
    pval < .001 ~ "< .001",
    pval < .01 ~ "< .01",
    pval < .05 ~ "< .05",
    TRUE ~ broman::myround(pval, 3)
  )
}

# round to nearest non-zero decimal
my_round = function(x, n=2) {
  max(abs(round(x, n)), abs(signif(x, 1))) * sign(x)
}
```

## Learning Objectives

Today we will learn...

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

## Workflow

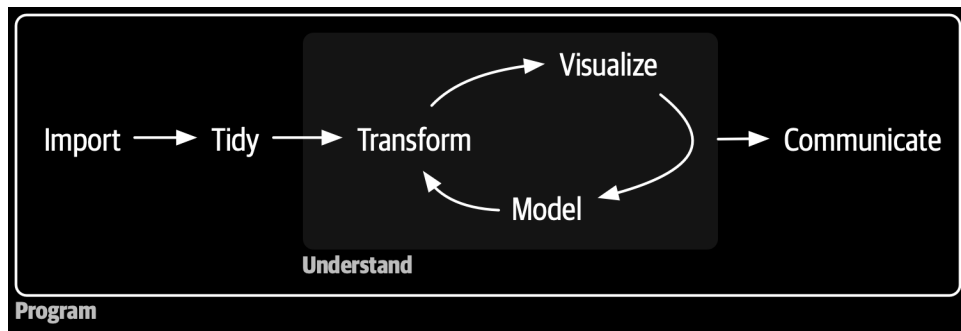


Figure 1: Image source: Wickham et al. (2023) (all rights reserved)

## Set-up environment

- always start with a clean R Environment (`Session > Restart R`)
  - go to the `Session >` select `Restart R`
  - or use the keyboard shortcut `:Cmd/Ctrl+Shift+0`

- 
- I usually run `options(scipen=999)` to suppress scientific notation

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

- load in our required packages

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

- and set my preferred `ggplot2` theme

```
# set ggplot theme
theme_set(theme_bw())
```

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

- $y \sim x$  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.
- our first model will be:

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

### Load data

- load our data using the `read_csv()` function from `readr`
- the `clean_names()` function from the `janitor` package 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()
```

### Mini-EDA

- Exploratory Data Analysis is usually first step once collecting data
  - involves plotting and summarising variables of interest
- let’s explore the data a little bit, which is what we would normally do before fitting any models

#### `head()`

- let’s use `head()` from base R to see the first 6 rows of our data

```
# 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
```

- 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

`summary()`

- `summary()` provides summaries of each variable in a dataframe
- numeric variables: descriptive statistics for the centre and spread of the data (mean, median, quartiles)
- categorical data: count per category
- character variables: number of observations

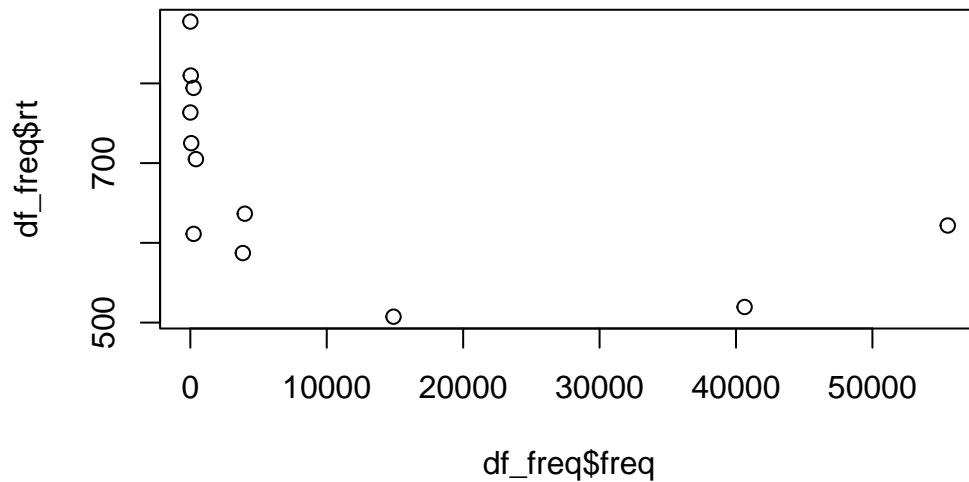
```
summary(df_freq)
```

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

## Plotting

- what does the relationship between `freq` and `rt` look like?

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



- a lot of frequency values below roughly 400
  - 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()

- `lm()` function fits simple linear models
- as arguments it takes a formula ( $y \sim x$ ) and a dataset, at minimum
- for now, we will use 1 for our predictor, which is a placeholder for the intercept

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

- 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

- to run such a model with our `df_freq` data:

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

- or, to save the model as an object so that we can call on it later, assign it a name (`name <- value`)

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

#### 💡 Object naming

- the letters `df` in `df_freq` stand for ‘data frame’
  - this serves as a reminder of what exactly that object in our environment is
- 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 (there are no rules)
- if we plot the frequency data, we could call save the plot as `fig_freq` or `plot_freq`
- this simply helps keep our environment structured, which will become useful when you begin having more objects in your environment at a time

### Model output

- print our model

```
# print model
fit_rt_1
```

Call:

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

Coefficients:

```
(Intercept)
      679.9
```

- `intercept` and `slope` are called **coefficients**
  - Why do we only see `Intercept`?
  - because we didn’t include any predictors in our model.

- 
- 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

#### 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()
```

## 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 Standard deviation of residuals/error in our model (lower = better)

## Intercept

- 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
```

- intercept corresponds to the mean reaction time value
  - why is this?

### Plotting $rt \sim 1$

- Figure ?? 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?

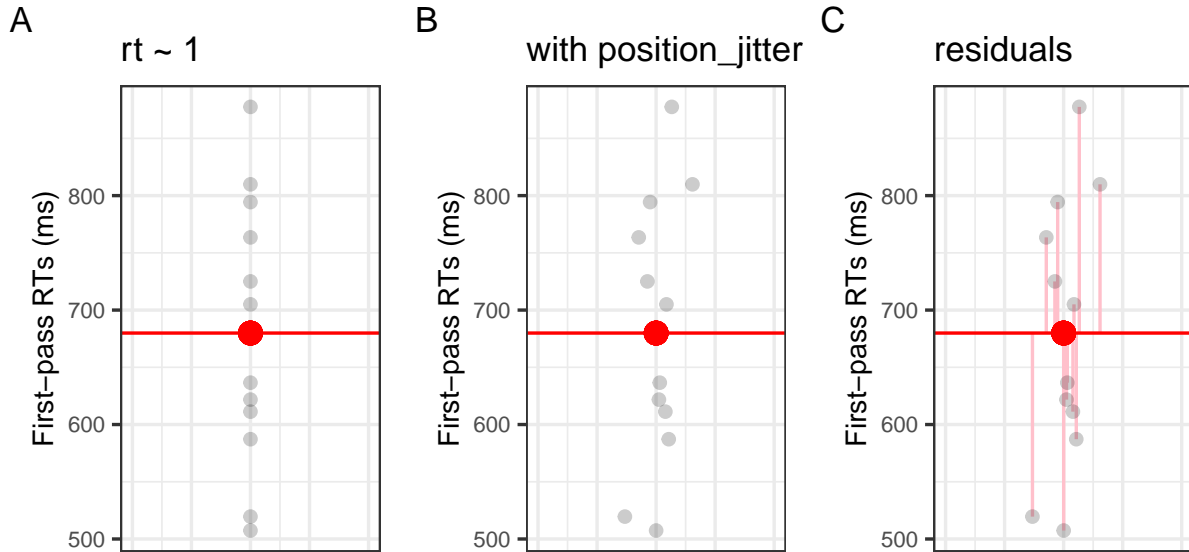


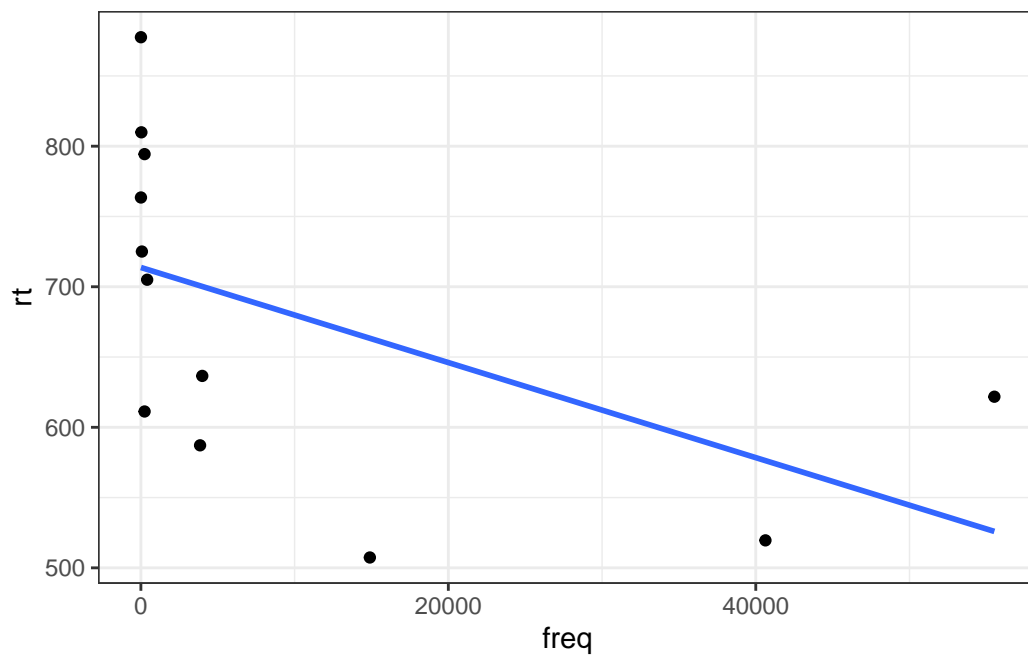
Figure 2: Visualisation of ' $rt \sim 1$ ': observed values (black) and mean (intercept; red). Residuals would be the distance from each black dot to the y-value of the red dot

## Adding a fixed effect (slope)

- 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`)
  - it tells us the *effect* our IV has on the DV (although be weary of making causal inferences)

- 
- let's first plot the data again, but with a line:

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



- what does this tell us?

## Fit model (treatment contrasts)

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

## 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

## Intercept

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

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

```
(Intercept)
713.7063
```

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

```
[1] 679.9167
```

### Slope

- the slope is -0.0033823
  - what does this correspond to?

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

```
freq  
-0.003382289
```

- change in  $y$  (our DV `rt`) for a 1-unit change in  $x$  (our IV: `freq`)
  - how we interpret this value depends on the measurement unit your variables are

### Exploring the model

- we can extract information from our model and compare it to our observed data

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

```
[1] 12
```

```
# how many fitted values does our model have?  
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: predicted values

- 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 predicted reaction time for a word with frequency score of 5000?

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

```
(Intercept)
696.7949
```

## Model assumptions

- is our model a good fit for our data?
- linear regression makes assumptions about our data
  - these assumptions relate to the *residuals* of our model, not the raw data points themselves
- we'll focus on two assumptions for now:
  - assumptions of *normality* of the residuals
  - the constant *variance* of the residuals
- both assumptions are often diagnosed visually, so it takes some practice to learn what looks right

### Normality

- a model's *residuals* (i.e., the difference between the *fitted* and *observed* values) should be approximately normally distributed
- Normality is typically visualised using a histogram (Figure ?? A) and/or a quantile-quantile (Q-Q) plot (Figure ?? B).

### Constant variance

- if 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
- typically visualised using a residual plot, which should look like a blob (Figure ?? C).

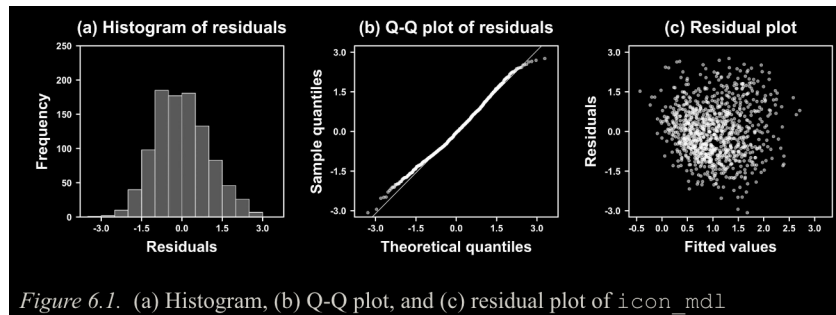


Figure 3: Image source: Winter (2019) (all rights reserved)

## Visualising model assumptions

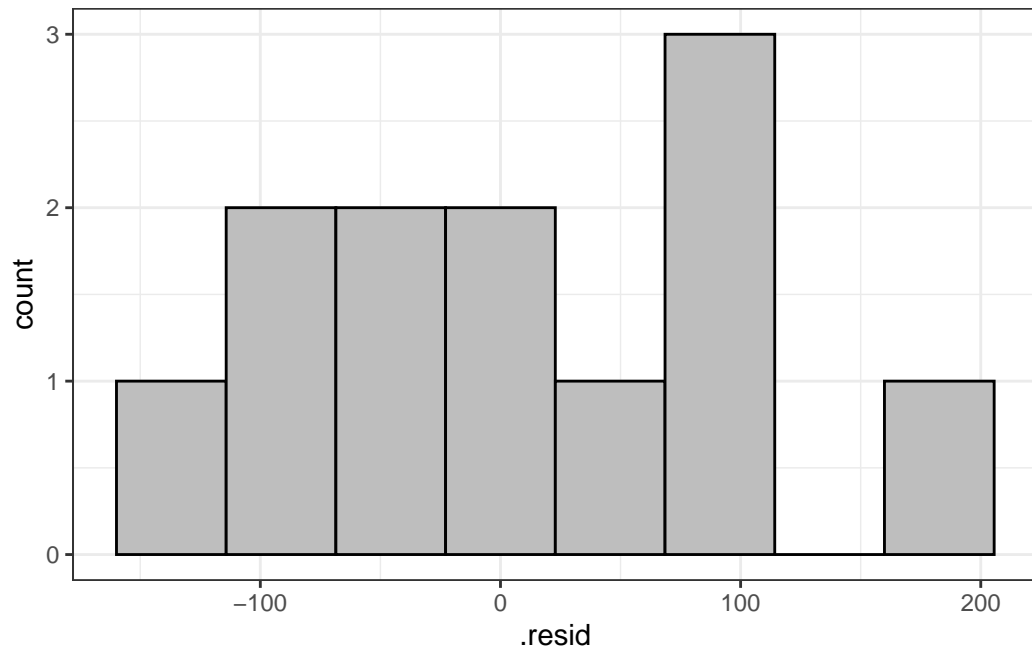
- let's plot our residuals to assess whether our model satisfies the assumptions of normality and constant variance

### Histogram

- let's 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()`).

```
df_freq <- broom::augment(fit_rt_freq, df_freq)

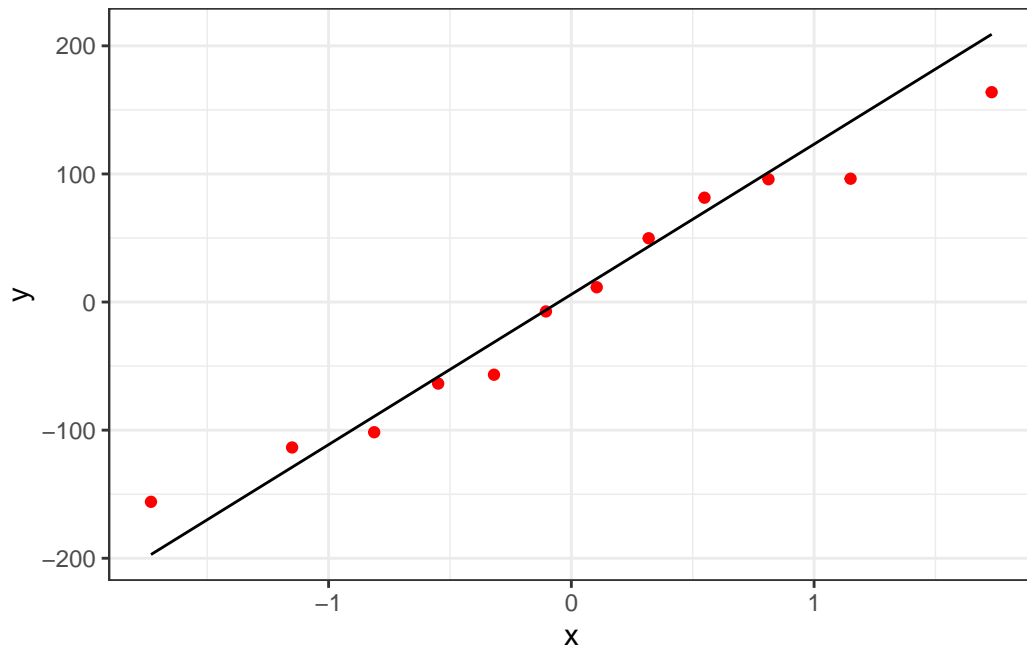
# and create ggplot
df_freq |>
  ggplot() +
  aes(x = .resid) +
  geom_histogram(bins = 8, fill = "grey", colour = "black") +
  theme_bw()
```



### Q-Q plot

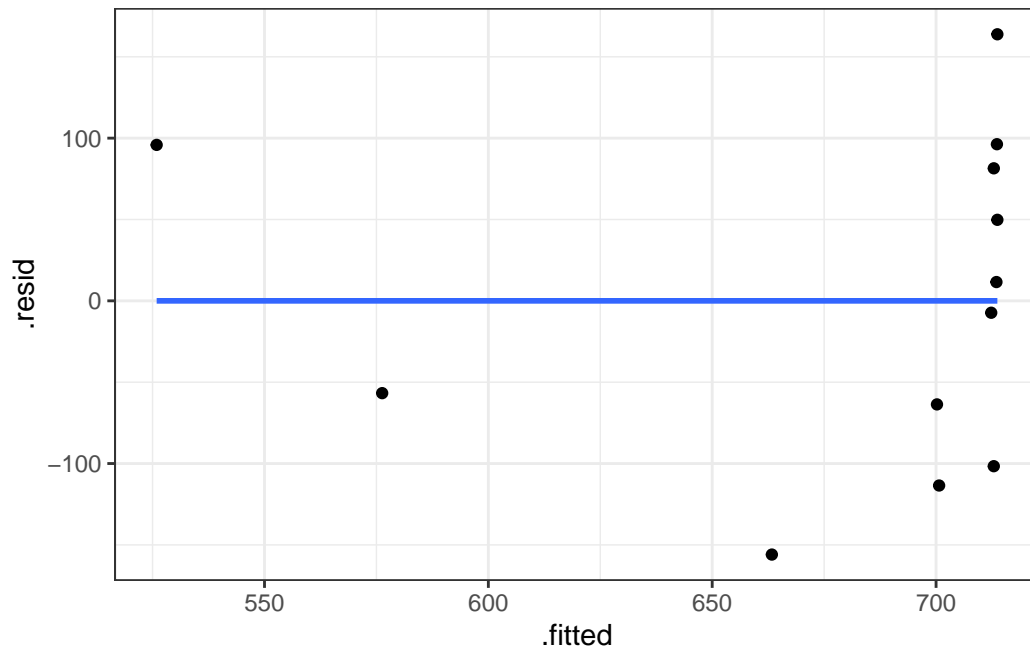
```
df_freq |>
  ggplot() +
  aes(sample = .resid) +
  geom_qq(colour = "red") +
  geom_qq_line()
```





### Residual plot

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



## Reporting our model

- following Sonderegger (2023) (Section 4.6.1), we should report
  - our individual coefficients (coefficient estimate, standard error, test statistic (e.g.,  $t$ -value) and corresponding  $p$ -value)
  - measures of model fit

## Coefficients

- this can be written

Higher-frequency words had longer reaction times, but this effect was not significant ( $\hat{\beta} = -0.003$ ;  $t = -1.99$ ).

- and/or presented in a table

```
# issue with mathmode in quarto; use unicode and fmt_markdown (https://kazuyanagimoto.com/
tidy(fit_rt_freq) |>
  mutate(p.value = format_pval(p.value),
         estimate = round(estimate,3),
         std.error = round(std.error,3),
```