Multiple regression

WiSe23/24

Daniela Palleschi

2023-10-11

Table of contents

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)
```

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

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

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)
```

- the variable `log10freq` is a remnant from Winter (2019); we'll use the natural logarithm

Multiple regression

- so far we've run simple linear models, which are equivalent to
  - a one-sample t-test (intercept-only model)
  - two-sample t-test (for a categorical predictor)
  - Pearson's r (for a standardised continuous predictor)
- why then should we bother with linear regression?
  - it allows us to include *multiple* predictors in our models simultaneously
    - still boils down to modeling the mean, but while conditioning the mean on multiple variables at once

Extending the equation of a line

- the equation of a line (Equation 1):
  - value of y = the intercept (b_0) + the corresponding value of x multiplied by the slope ($b_1 x$) + the error (residuals (e))
- In multiple regression, we can include more than one slope (Equation 2)

$$y = b_0 + b_1 x + e \tag{1}$$

$$y = b_0 + b_1 x + b_2 x + ... + e \tag{2}$$

One predictor

- re-run our simple model with this dataset
  - keep reaction times un-transformed for now

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

```
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
```

- there is a decrease in reaction times (-37.5 milliseconds) for a 1-unit increase in log frequency

---

- model fit using `glance()`:

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

```
[1] 0.3834186
```

- our model describes 38% of the variance in response times
- is this described variance due solely to frequency?
  - Other effects that are correlated with frequency might be conflating the frequency effect
  - Let's expand our model to include word length (Equation 3)

$$y = b_0 + (b_1 * log\ frequency) + (b_2 * word\ length) \tag{3}$$

Adding another predictor

- add `length` as a predictor

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

```
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
```

- an increase in word length (+1 letter) corresponds to a 20ms increase in reaction times
  - our intercept is now 748ms, instead of 907ms
  - this corresponds to the prediction for reaction times to a word with 0 log frequency and 0 word length, but this is not very interpretable

Intercept = y when x_1 and $x_2 = 0$

- centering both predictors would result in an intercept reflecting the reaction time for a word with average frequency and average length (because 0 for each would equal the mean)

```
b0 <- tidy(fit_freq_mult)$estimate[1] # intercept estimate
b1_freq <- tidy(fit_freq_mult)$estimate[2] # freq estimate
b1_length <- tidy(fit_freq_mult)$estimate[3] # length estimate
```

```
b0 + b1_freq*0 + b1_length*0
```

```
[1] 748.4261
```

Change in slope

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

```
# simple model (rt ~ freq)
tidy(fit_freq_full) |> select(term, estimate)
```

```
# A tibble: 2 x 2
  term        estimate
  <chr>          <dbl>
1 (Intercept)    907.
2 log(freq)      -37.5
```

```
# mult reg model (rt ~ freq + length)
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
```

Change in R^2

- including `length` increases the variance described by our model, reflected in the R^2 values (0.49 instead of 0.38

```
# simple model (rt ~ freq)
glance(fit_freq_full)$r.squared
```

```
[1] 0.3834186
```

```
# mult reg model (rt ~ freq + length)
glance(fit_freq_mult)$r.squared
```

```
[1] 0.4872977
```

Standardising our predictors

- when we have multiple continuous predictors, standardising them can help their interpretation
  - because their slopes are comparable
- we can do this by centering each variable and then dividing by the standard deviation (like yesterday), or we could use the `scale()` function, which has the same result

```
# 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
```

Model with z-scored predictors

- use `scale()` for `freq` and `length`.

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

- fit a model with them as predictors

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

Inspect model: R^2

- let's check the R^2:

```
# mult reg model: (rt ~ freq + length)
glance(fit_freq_mult)$r.squared
```

```
[1] 0.4872977
```

```
# z-scored mult reg model: (rt ~ freq_log_z + length_z)
glance(fit_freq_z)$r.squared
```

```
[1] 0.4872977
```

- $R^2 = 0.49$, just like above

  - this is a reminder that the predictors still represent the same variance in the underlying model
  - their units and scales have simply changed

Inspect model: coefficient estimates

```
# rt ~ freq + length
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
```

```
# rt ~ freq_log_z + length_z
tidy(fit_freq_z) |> select(term, estimate)
```

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

- a 1-unit change now corresponds to a change of 1 standard deviation (because we standardized, producing z-scores)
- frequency now has a larger magnitude than the effect of length
  - a 1-SD increase in frequency (holding length constant) = decrease in reaction times by 60.6 ms
  - a 1-SD increase in length (holding frequency constant) = increase in reaction times by 43.3 ms
  - does frequency influence the effect of length, and vice versa?

Adding an interaction term

- please check out Ch. 8 (Interations and nonlinear effects) in Winter (2019) for a more in-depth discussion on interactions

- 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 interaction terms of two predictors by connecting them with a colon (`:`).

```
lm(rt ~ freq_log_z + length_z + freq_log_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_log_z             -63.9
3 length_z                41.8
4 freq_log_z:length_z    -11.4
```

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

```
lm(rt ~ freq_log_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_log_z             -63.9
3 length_z                41.8
4 freq_log_z:length_z    -11.4
```

- 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

Model assumptions

- we've discussed the assumptions of normality and homoscedasticity (constant variance), which both refer to the residuals of a model
  - We typically assess these assumptions visually

Normality and Homoscedasticity

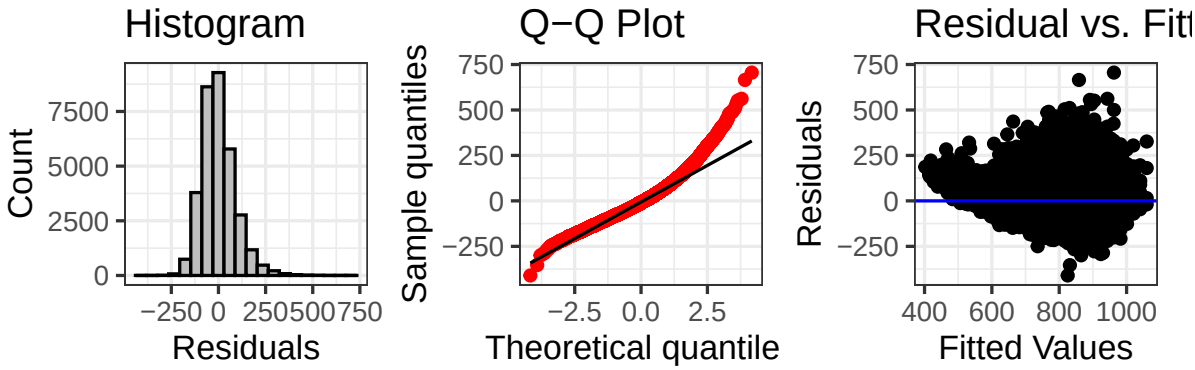
- for our model:

```
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
```



- not very reassuring, let's try log-transformed reaction times.

Log-transformed response variable

```
fit_freq_log_z <-
  lm(log(rt) ~ freq_log_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_log_z           -0.0826
3 length_z              0.0524
4 freq_log_z:length_z -0.00779
```

- our coefficients are much smaller, because they're on the log-scale

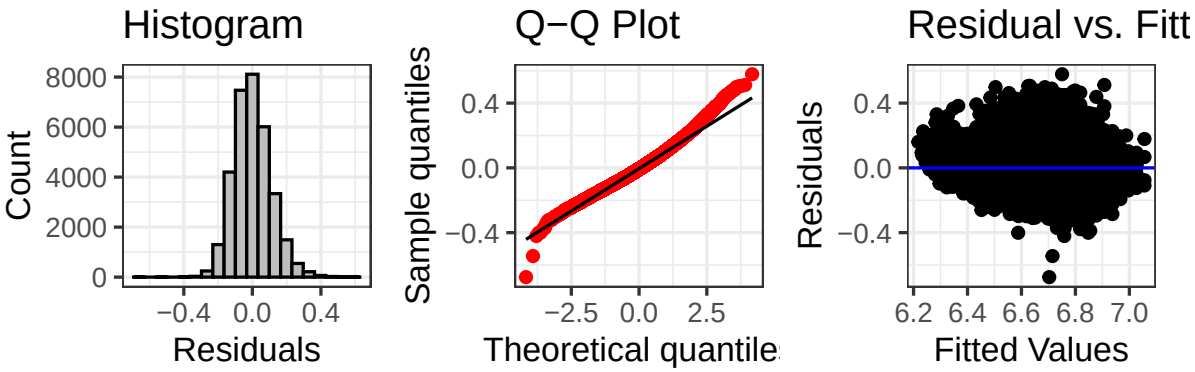
---

```
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 better

Collinearity

- collinearity refers to when continuous predictor variables are correlated
  - 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 correlated.

- the `vif()` function from the `car` package assesses collinearity, comparing *variance inflation factors*
- VIF values close to 1 indicates there is not a high degree of collinearity between your variables
  - higher than 1 indicates correlation; above 10 is highly correlated (thresholds may be field-specific)

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

```
      freq_log_z         length_z freq_log_z:length_z
        1.246509         1.184641            1.068283
```

- collinearity is a conceptual problem, and should be considered in the planning stage
- 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
- we should have a principled approach to model building and variable selection
- of course, we can add predictors in exploratory analyses, but this should always be reported as exploratory
  - and any observed effects replicated where possible

Adjusted R^2

- adjusted R^2 is a more conservative version of R^2 that takes into account the number of predictors in a model
- adjusted R^2 includes the number of predictors (k) in its denominator (bottom half of a division)
  - the more predictors there are, the smaller adjusted R^2 will be, unless each additional predictor explains sufficient variance to counteract this penalisation

---

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

```
[1] 0.4872667
```

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

```
[1] 0.5176475
```

- there is a small increase in adjusted R^2 when we include length and its interaction with frequency
  - this suggests that including `length` and the interaction term does not result in overfitting
  - i.e., length contributes to the variance explained by the model

---

- if we 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

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

```
[1] 0.5150461
```

Term	Definition	Equation/Code
NA	NA	NA

Important terms

Learning Objectives

Today we learned...

- 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

Task

Follow the instructions on the website (Multiple regression > Task) (or continue to the next slides)

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?

Literaturverzeichnis

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

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