Tutorial: logistic regression as a GLM

Andi Fugard, adapted by Marju Kaps

30 Nov 2021

1 Introduction

By the end of this week you will understand:

- How generalised linear models (GLMs) generalise regression to deal with a wider variety of distributions of data.
- How to fit a GLM to binary data (also known as logistic regression).
- How to interpret the results.
- Which diagnostics from linear models do and don't apply here.

The main challenge in going from linear regression to logistic regression is that the outcome variable is now on the log-odds scale, which makes it trickier to interpret the model intercept and slopes. I provide four different ways to make sense of them, so you can choose which makes most sense to you.

2 Setup

Let's load the `tidyverse`.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyver
se 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.5     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_con
flicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

I'm also going to use `knitr` to make tables look prettier:

```
library(knitr)
```

3 The dataset

We will look at a different dataset this week, as analysed by Mroz (1987), spotted in Fox and Weisberg (2019). The data are from the US Panel Study of Income Dynamics (PSID) in 1975: 753 married white women aged between 30 and 60.

The outcome variable we will examine is whether participants participated in the labour force. So it will be worth thinking through your (implicit or otherwise) theories of what processes might be operating in 1975, and if similar processes are expected to apply in 2021.

Variable name	Description
`lfp`	labour-force participation at some point during the year (no/yes).
`lfp_yes`	same as above, but with yes = 1, no = 0
`k5`	number of children 5 years old or younger.
`k618`	number of children 6 to 18 years old.
`age`	in years.
`wc`	wife's college attendance (no/yes).

Variable name	Description
`hc`	husband's college attendance (no/yes).
`lwg`	log expected wage rate; for women in the labour force, the actual wage rate; for women not in the labour force, an imputed value based on the regression of `lwg` on the other variables.
`inc`	family income exclusive of wife's income (in $1000s).

Read in the data:

```
dat <- read.csv("psid1975.csv")
```

Here is how to view a random sample of rows. (The `set.seed` command is optional – I am just using it so that I keep getting the same random sample every time I run the code.

```
set.seed(3)
dat %>%
  slice_sample(n = 10)
```

lfp <chr>	lfp_yes <int>	k5 <int>	k618 <int>	age <int>	wc <chr>	hc <chr>	lwg <dbl>	inc <dbl>
no	0	0	2	43	no	no	1.2282799	14.550
no	0	1	1	30	no	yes	0.9791328	22.000
no	0	0	0	53	no	no	0.8861951	21.574
no	0	0	3	39	no	no	0.8532125	28.363
yes	1	0	2	35	yes	no	1.6046872	5.120
yes	1	0	2	36	yes	yes	1.7037485	23.600
yes	1	0	2	48	no	no	1.5448993	9.000
no	0	0	0	50	yes	yes	1.7312460	63.200
no	0	1	0	30	no	yes	1.0167637	19.392
yes	1	0	1	48	yes	yes	1.6549002	70.750

1-10 of 10 rows

The `lfp_yes` variable was calculated using the line below. This might come in handy for your projects should you wish to dichotomise a variable.

```
as.numeric(dat$lfp == "yes")
```

4 Warmup activity

4.1 Activity | 4.2 Answer

a. What do you think might predict participation in the labour force? Have a think *before looking at the descriptives* and write down your thoughts.
b. Set `wc` and `hc` to factors.
c. Generate a table of descriptive statistics ("Table 1") using a method of your choosing.

5 Fitting a logistic regression model in R

We will spend some time going into the arithmetic of logistic regression model coefficients since it helps interpret what the models mean.

So that you have a sense of where this is heading, here is how to fit a model predicting whether women were in paid employment from the number of children in the family aged 5 or younger (`k5`) and 6 to 18 (`k618`):

```
mod_kids <- glm(lfp_yes ~ k5 + k618, data = dat, family = binomial)
```

This is the intercept-only model:

```
mod_0 <- glm(lfp_yes ~ 1, data = dat, family = binomial)
```

And this command compares the two:

```
anova(mod_0, mod_kids, test = "Chi")
```

To get a summary use:

```
summary(mod_kids)
```

Here are confidence intervals:

```
library(car)
Confint(mod_kids)
```

The R code involved is almost identical to what we used for linear regression. The main challenge will be interpreting the slopes, but I will share some tips to make this relatively painless.

6 What is a generalised linear model?

As the name suggests, *generalised linear models* generalise the linear model. (Not to be confused with a *general* linear model – different thing.) They are initialised as GLM, which is pronounced "glim".

GLMs have three components (see Fox and Weisberg, 2019, pp. 272-275 for more detail):

1. A specification of the *error distribution*. For linear models, this is a normal distribution.
2. A linear formula linking outcome to predictors and the β's to be estimated:

   $$g(y) = \beta_0 + \beta_1 x_1 + \beta_1 x_2 + \cdots + \beta_n x_n$$

   The right hand side is identical to linear models. The left hand side, $g(y)$, is mildly different to the usual y and is there to ensure the maths works for different shapes of distribution.
3. GLMs also have a *link function* – that's the g. For linear models, $g(y) = y$, what is known as the *identity link* – it just passes the input unchanged to the output. For logistic regression, g is log-odds, also known as the *logit*. The formula then is

   $$\log \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_1 x_2 + \cdots + \beta_n x_n$$

   where p is the probability of an event happening (e.g., being in work).

So, a linear regression model is a generalised linear model with a normal error distribution and an identity link. Logistic regression is a generalised linear model with a binomial error distribution and a log-odds (logit) link.

The error distribution and link are together called the *family*. By default, `glm` chooses a sensible link for you.

7 What is the log function?

To interpret logistic regression models we need to understand log-odds and to understand log-odds we need to grasp logs. I'm aware that it may be some time since you last used them!

There are two aspects to revise: the arithmetic and why logs are typically used.

7.1 The arithmetic

The arithmetic is easy: the *log* is simply the inverse of "to the power of" or exponentiation.

Here is an example of exponentiation:

$$10^n = \underbrace{10 \times \cdots \times 10}_{\text{n times}}$$

So $10^2 = 100$ and $10^4 = 10000$ and so on. More generally, with b^n, the b is called the *base* and n is the *exponent*.

Log just reverses this.

Suppose with want to find out what to raise 10 to the power of to get 1000, then we just calculate

$$\log_{10}(100)$$

In R:

```
log(100, base = 10)
```

```
## [1] 2
```

So $10^2 = 100$ and $\log_{10}(100) = 2$.

7.1.1 Activity | 7.1.2 Answer

Two to the power of what is 1024...?

7.2 Why log?

Sadly, these days we are very aware (https://www.weforum.org/agenda/2020/04/covid-19-spread-logarithmic-graph/) of exponential and log functions since they are used when communicating data on the Covid-19 pandemic.

Suppose some quantity doubles over time; let's take as an example the number of cat GIFs in my saved images over time:

```
doubles <- tibble(month    = 0:9,
                  cat_GIFs = 2^month)
doubles %>%
  kable(col.names = c("Month", "Number of cat GIFs"))
```

Month	Number of cat GIFs
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512

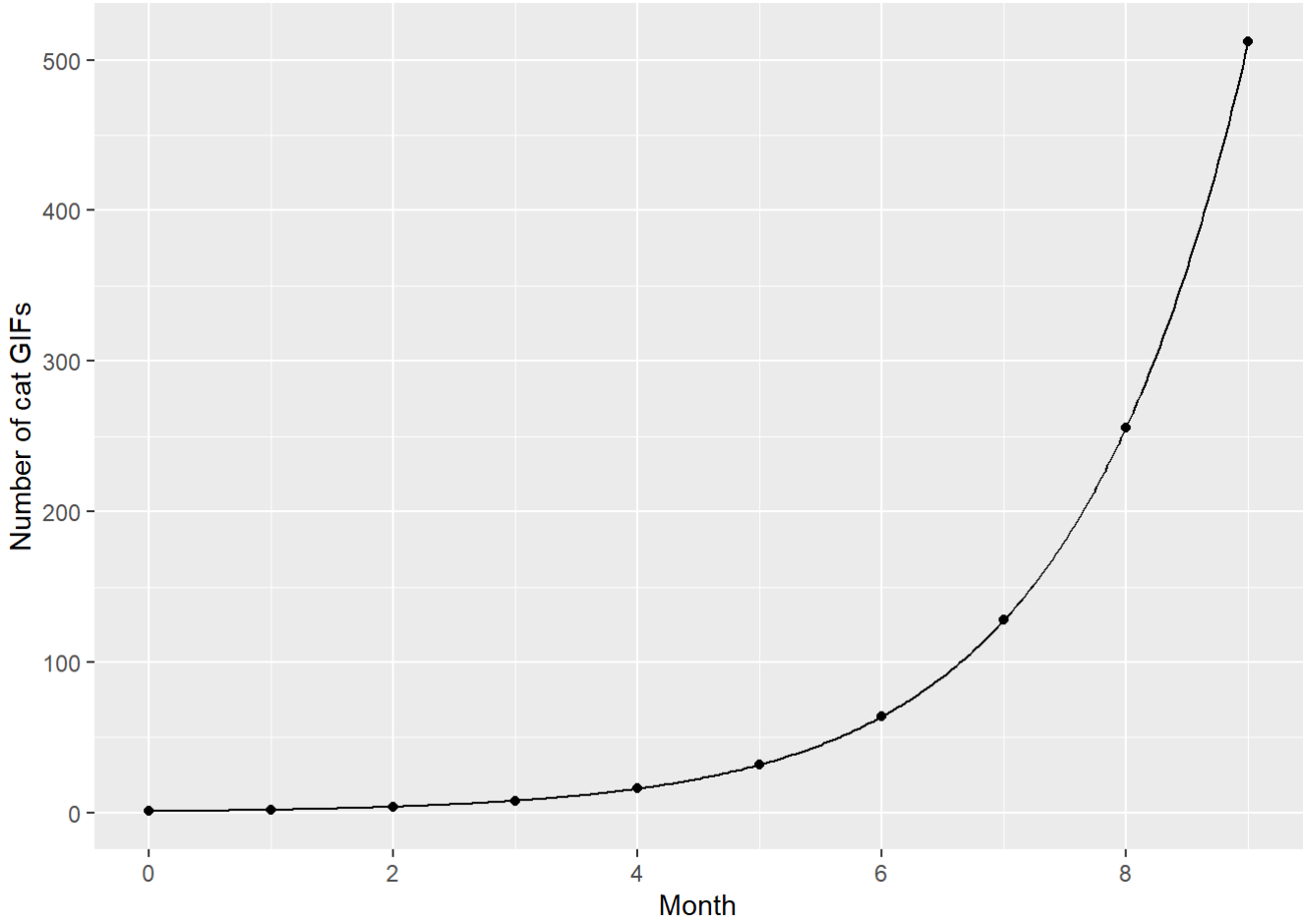
I started off with one GIF. As each month passes, the number of GIFs doubles. To find the number of GIFs at a particular time, use 2^{month}. Here is 5 months into my stash:

```
2^5
```

```
## [1] 32
```

Here is a picture of GIF growth:

```
doubles %>%
  ggplot(aes(month, cat_GIFs)) + geom_point() +
  labs(x = "Month", y = "Number of cat GIFs") +
  scale_x_continuous(breaks = seq(0,9,2)) +
  geom_function(fun = function(x) 2^x, n = 400)
```



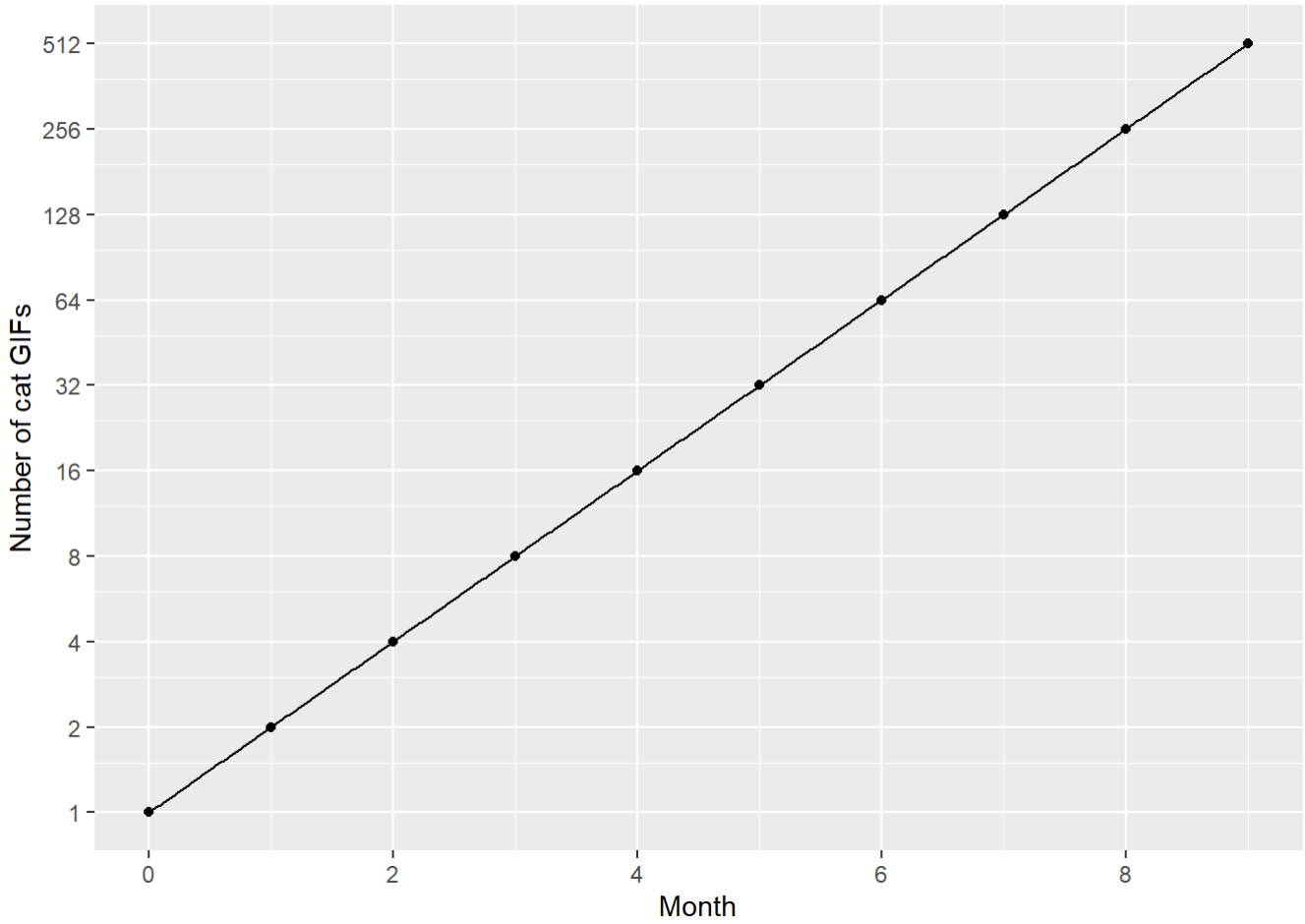
As we saw above, we can also run 2^{month} in reverse. Given a particular number of cat GIFs, say 32, how many times did the original number double to get there? Simply calculate $\log_2(32)$:

```
log(32, base = 2)
```

```
## [1] 5
```

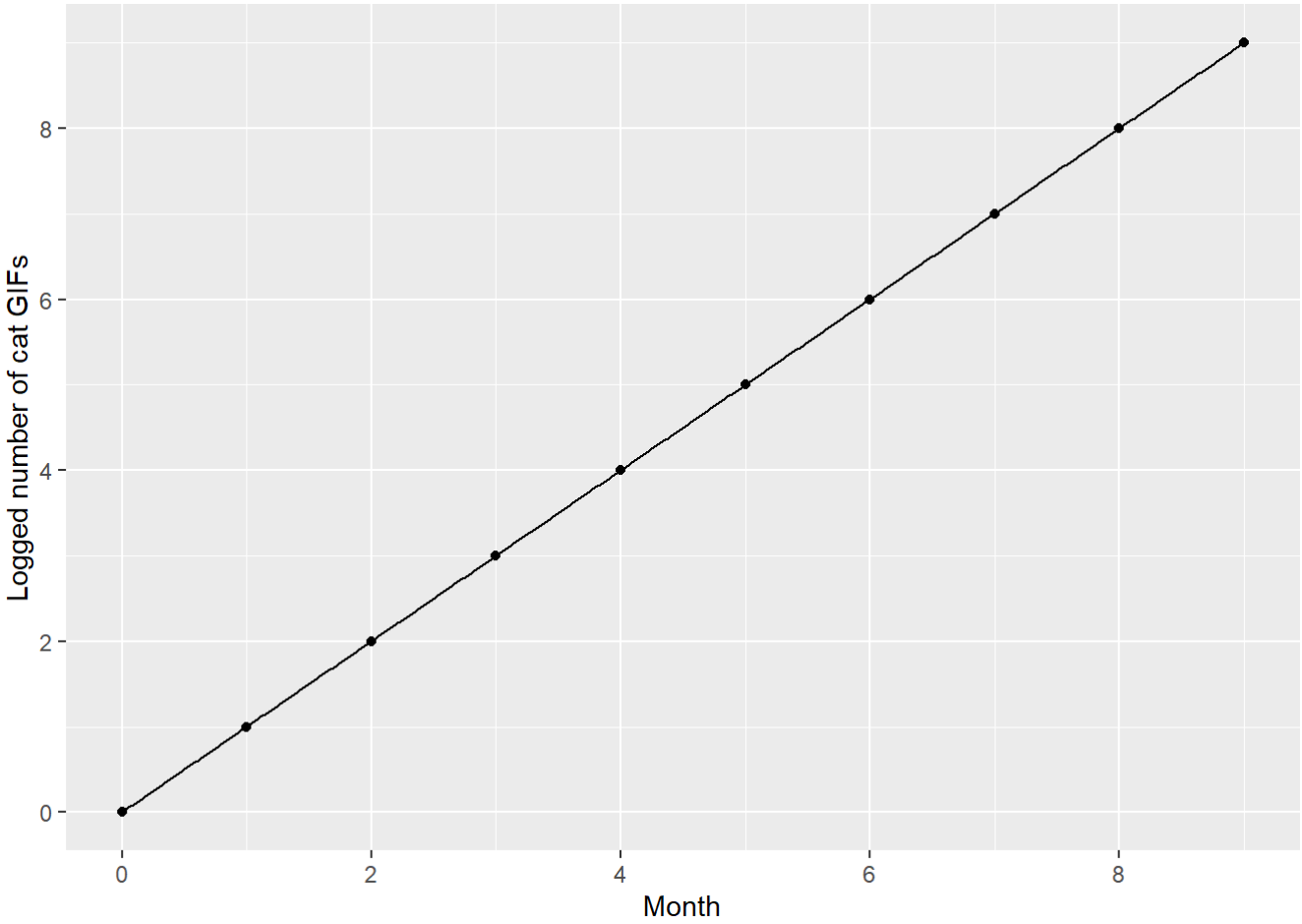
Often it is easier to see exponential relationships and any changes in relationships over time if they are plotted on a straight line. A way to straighten out exponential growth is to log the y-axis whilst keeping the original values at each tick point on the axis:

```
doubles %>%
  ggplot(aes(month, cat_GIFs)) + geom_point() +
  coord_trans(y = "log2") + # this line transforms the axis
  labs(x = "Month", y = "Number of cat GIFs") +
  scale_x_continuous(breaks = seq(0,9,2)) +
  scale_y_continuous(breaks = 2^seq(0,9,1)) +
     # this scale_y line says where to place the labels
  geom_function(fun = function(x) 2^x, n = 400)
```



We can also log the values, so that the y-axis now says how many times the number of GIFs has doubled.

```
doubles %>%
  ggplot(aes(month, log(cat_GIFs,2))) + geom_point() +
  labs(x = "Month", y = "Logged number of cat GIFs") +
  scale_y_continuous(breaks = seq(0,10,2)) +
  scale_x_continuous(breaks = seq(0,9,2)) +
  geom_function(fun = function(x) x, n = 400)
```



I hope that provides some intuitions for what the log function does. Roughly, it transforms a process that multiplies, and can be hard to understand on the original scale, into a process that is additive.

8 Intercept-only models again

For linear regression, the intercept in an intercept-only model was the mean of the the outcome variable. Intercepts in GLMs work the same; however, now we have the mild complication of the link function which makes GLMs work.

The mean of the binary variable, `lfp_yes` , is:

```
mean(dat$lfp_yes)
```

```
## [1] 0.5683931
```

This is the proportion of women who were in paid employment (do you see why the mean is a proportion here?).

```
table(dat$lfp)
```

```
##
##  no yes
## 325 428
```

Now let's see what logistic regression gives us:

```
mod_intercept <- glm(lfp_yes ~ 1, data = dat, family = binomial)
summary(mod_intercept)
```

```
##
## Call:
## glm(formula = lfp_yes ~ 1, family = binomial, data = dat)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.296  -1.296   1.063   1.063   1.063
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.27530    0.07358   3.742 0.000183 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1029.7  on 752  degrees of freedom
## Residual deviance: 1029.7  on 752  degrees of freedom
## AIC: 1031.7
##
## Number of Fisher Scoring iterations: 4
```

Most of that will look very familiar – the coefficients, standard error, p-values (though now of a z-value rather than t) – however, the intercept is *not* the mean of `lfp_yes`.

```
coef(mod_intercept)
```

```
## (Intercept)
##    0.275298
```

This is actually the *log-odds* of being in paid work, which is arithmetically related to the probability of being in paid work.

Let's see how this works below.

9 Odds and log odds

Odds are alternatives to probabilities for quantifying uncertainty. The probability of getting a 20 on one roll of a fair 20-sided die are $\frac{1}{20}$ or 0.05. The odds are 1 to 19 or $\frac{1}{19}$.

There are 20 possible outcomes, each equally likely. We are interested in the odds of one of those outcomes. The odds are the ratio of the probability that an event *will* happen to the probability that it will *not*. For the 20-sided die case, since every outcome is equally likely, we can just use the number of outcomes directly in the formula.

More generally, if the probability of an event is p, then the odds are

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

If the probability of an event happening is p, then $1-p$ is the probability that it will not happen, hence the bottom line of that fraction.

It is also possible to do the conversion in reverse:

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

The log odds are the log of the odds, usually the natural log (https://en.wikipedia.org/wiki/Natural_logarithm), i.e., log base $e = 2.718282\ldots$, which is what R provides by default. This can be written $\log_e$ or ln. So the log odds of getting a 20 on one throw of a 20-sided die are:

```
log(1/19)
```

```
## [1] -2.944439
```

Personally, I don't find -2.94 a particularly intuitively interpretable number! And actually, later I will share some tricks so that you don't have to even think about what it means. However, I think it is important to get a sense of some example landmark odds and log odds along a range of probabilities. A few minutes spent staring at this table will come in handy (`Inf` is "infinity").

Probability	Odds	Log odds (logit)
0.0	0.00	-Inf
0.1	0.11	-2.20
0.2	0.25	-1.39
0.3	0.43	-0.85

Probability	Odds	Log odds (logit)
0.4	0.67	-0.41
0.5	1.00	0.00
0.6	1.50	0.41
0.7	2.33	0.85
0.8	4.00	1.39
0.9	9.00	2.20
1.0	Inf	Inf

Here are clues for where to look:

- Look at what the odds and log odds are when the probability is 0.5.
- Look at probabilities 0 and 1.
- Look for symmetry in the log odds above and below probability 0.5.
- The odds are also symmetric around 0.5, though this is a (little) more difficult to see. Try 0.2 and 0.8 probability. The odds of a 0.2 probability are

$$\frac{0.2}{1-0.2} = \frac{0.2}{0.8} = \frac{2}{8} = \frac{1}{4} = 0.25$$

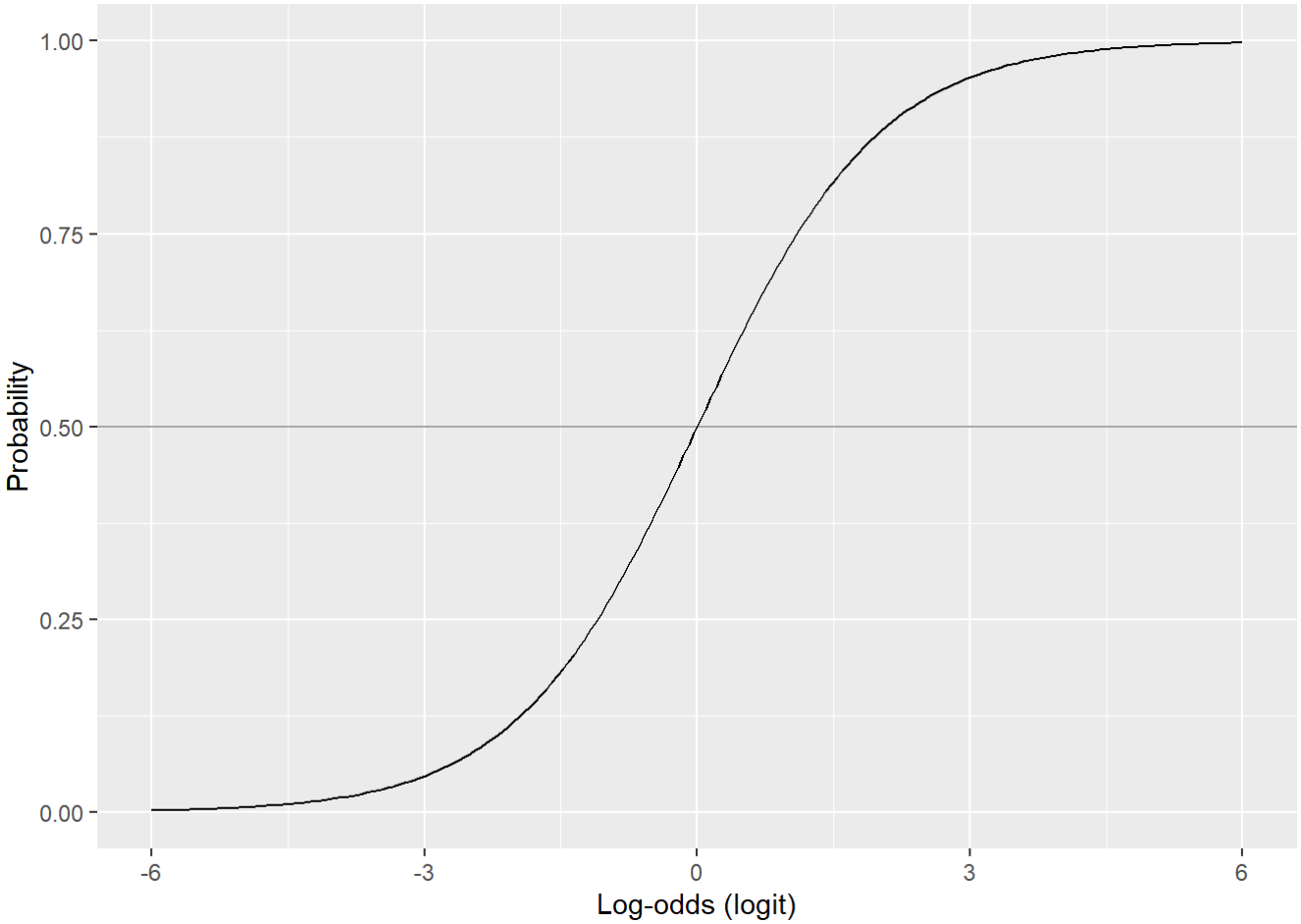
  The odds of a 0.8 probability are

$$\frac{0.8}{1-0.8} = \frac{0.8}{0.2} = \frac{8}{2} = \frac{4}{1} = 4$$

  So the fraction just turns upside down.

Finally, here is a picture showing the relationship between log-odds and probability:

```
ggplot() +
  xlim(-6, 6) +
  geom_hline(yintercept = 0.5, colour = "darkgrey") +
  geom_function(fun = plogis, n = 200) +
  ylab("Probability") +
  xlab("Log-odds (logit)")
```



Note how the relationship is mostly non-linear, except for a patch in the middle where it is linear. The log-odds stretch out to negative and plus infinity, whereas probabilities are bound between 0 and 1. The probability for a log-odds of 6 is 0.9975274. The probability for a log-odds of 15, far off the scale on this graph, is 0.9999939. It never quite reaches 1 or, in the opposite direction, 0.

To summarise then, probability, odds, and log-odds are different and interchangeable ways to quantify how sure you are that something is going to happen.

10 Back to that intercept

We know the proportion of women in paid work:

```
mean(dat$lfp_yes)
```

```
## [1] 0.5683931
```

Assuming a random sample, this is also an estimate of the probability that someone chosen at random from the larger population will be in paid work. Let's then calculate the odds:

```
the_probability <- mean(dat$lfp_yes)
the_odds <- the_probability / (1-the_probability)
the_odds
```

```
## [1] 1.316923
```

Now the log-odds:

```
log(the_odds)
```

```
## [1] 0.275298
```

Back to the model coefficient:

```
coef(mod_intercept)
```

```
## (Intercept)
##    0.275298
```

HURRAH!

We could also go in reverse from the intercept. The function `exp(x)` calculates e^x which is the inverse of $\log_e$. This gives us the odds again:

```
the_odds_again <- exp(coef(mod_intercept))
the_odds_again
```

```
## (Intercept)
##    1.316923
```

Now to get the probability:

```
the_odds_again / (1+the_odds_again)
```

```
## (Intercept)
##   0.5683931
```

$$HURRAH^{HURRAH}!$$

Now we have got the hang of intercept-only models, onto slopes...

11 Interpreting model slopes

Look again at the graph:

Our hindsight hypothesis (having looked at the descriptive statistics in Table 1) – not a good way to do science, but fine for pedagogy – is that the number of children aged 5 or under will predict labour-force participation, but the number aged 6 or over will be irrelevant.

```
mod_children <- glm(lfp_yes ~ k5 + k618,
                    data = dat,
                    family = binomial)
summary(mod_children)
```

```
##
## Call:
## glm(formula = lfp_yes ~ k5 + k618, family = binomial, data = dat)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4566  -1.3712   0.9634   0.9953   1.7617
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.44479    0.11106   4.005 6.20e-05 ***
## k5          -0.87926    0.15817  -5.559 2.71e-08 ***
## k618         0.02730    0.05767   0.473    0.636
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1029.75  on 752  degrees of freedom
## Residual deviance:  994.53  on 750  degrees of freedom
## AIC: 1000.5
##
## Number of Fisher Scoring iterations: 4
```

And indeed that is what we get. But how should we interpret those coefficients? Here is a suite of options. Any one of these would and has been publishable (see if you can spot them in the literature), but they are not equally easy to interpret.

11.1 Interpret on the log-odds scale

This summary is a linear model on the log-odds (i.e., logit) scale. So we can interpret as before, just always ensuring we remember that the outcome units are log-odds rather than probabilities.

11.1.1 Activity | 11.1.2 Answer

Have a go at interpreting the intercept and slopes, focussing on the direction of effects and whether they are statistically significant.

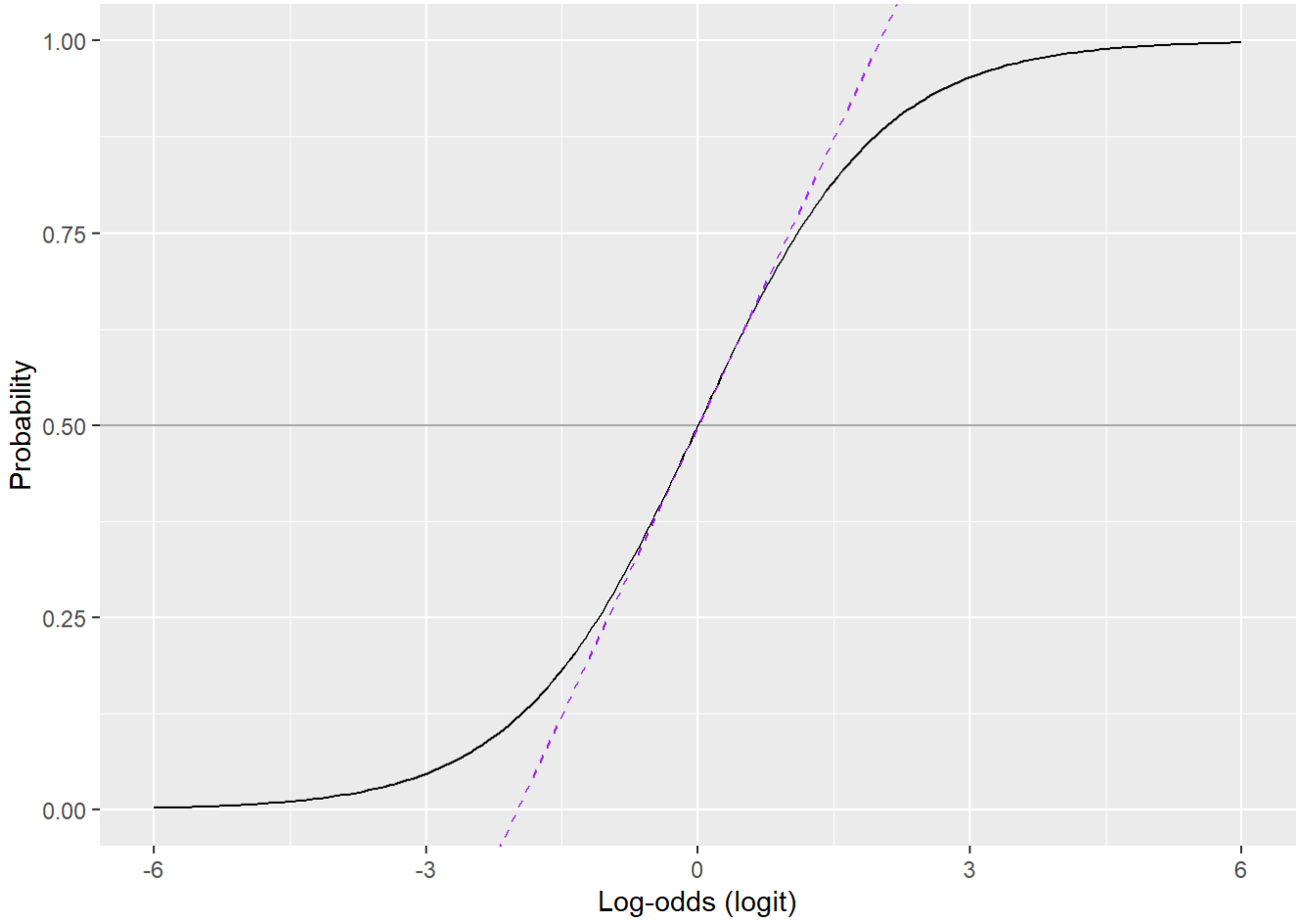
11.2 Interpret using the "divide-by-4" approximation

Gelman and Hill (2007, p. 82), provide a handy approximation for transforming slopes on the log-odds scale to changes in probability:

> "As a rule of convenience, we can **take logistic regression coefficients (other than the constant term) and divide them by 4 to get an upper bound of the predictive difference corresponding to a unit difference in x**. This upper bound is a reasonable approximation near the midpoint of the logistic curve, where probabilities are close to 0.5."

(Bold emphasis added.)

Here is the rule in a picture:



Where the the dashed purple line overlaps the curve, the probability $p = 0.5 + \frac{logit}{4}$.

11.2.1 Activity | 11.2.2 Answer

Can you decode that paragraph from Gelman and Hill and use it to interpret the coefficients?

```
coef(mod_children)
```

```
## (Intercept)          k5        k618
##  0.44478849 -0.87925889  0.02729833
```

11.3 Interpret using odds

There is an easy way to transform the predictors so that interpretation are on the odds scale, which are more interpretable than log-odds. Let's explore why it works – skip to the So What? section on a first read if you wish.

11.3.1 Some arithmetic

Here is the model formula again:

$$\log_e \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_1 x_2 + \cdots + \beta_n x_n$$

The left hand side gives the log odds and the right hand side is the linear formula we know and love from linear regression.

If we exponentiate both sides, this removes the log on the left and, er, complicates the right:

$$\frac{p}{1-p} = e^{\beta_0 + \beta_1 x_1 + \beta_1 x_2 + \cdots + \beta_n x_n}$$

The left hand size now gives the odds. We can simplify the right to:

$$\frac{p}{1-p} = e^{\beta_0} \times e^{\beta_1 x_1} \times e^{\beta_1 x_2} \times \cdots \times e^{\beta_n x_n}$$

since $b^{x+y} = b^x b^y$. Note how all the additions have become multiplications.

Since $b^{xy} = (b^x)^y$ we can separate out the exponentiated slopes, which is useful because it is very easy to get these from a `glm` model summary (by `exp` ing the slopes).

$$\frac{p}{1-p} = e^{\beta_0} \times (e^{\beta_1})^{x_1} \times (e^{\beta_2})^{x_2} \times \cdots \times (e^{\beta_n})^{x_n}$$

This is much easier to read if we rewrite e^x as $\exp(x)$:

$$\frac{p}{1-p} = \exp(\beta_0) \times \exp(\beta_1)^{x_1} \times \exp(\beta_1)^{x_2} \times \cdots \times \exp(\beta_n)^{x_n}$$

11.3.2 What does this mean? (Or: so what?)

Here are the coefficients from our model predicting being in paid work from number of children:

```
coef(mod_children)
```

```
## (Intercept)          k5        k618
##  0.44478849 -0.87925889  0.02729833
```

We can exponentiate (and round) them to:

```
exp(coef(mod_children))
```

```
## (Intercept)          k5        k618
##   1.5601602   0.4150904   1.0276743
```

The coefficients for the slopes are called **odds ratios**.

Slot them back into the formula:

$$\frac{p}{1-p} = 1.56 \times 0.42^{\texttt{k5}} \times 1.03^{\texttt{k618}}$$

This gives us an interpretation of the (exponentiated) coefficients in terms of odds.

For every increase in `k5` (children aged 0 to 5) by one child, the odds of being in work multiply by 0.42; in other words they are $100(1 - 0.42) = 58\%$ lower.

Let's try plugging some numbers in, first with no children:

$$\begin{aligned} \frac{p}{1-p} &= 1.56 \times 0.42^0 \times 1.03^0 \\ &= 1.56 \times 1 \times 1 \\ &= 1.56 \end{aligned}$$

So the odds of being in work if you have no children are 1.56 – i.e., the same as the exponentiated intercept.

Let's try one child aged 0 to 5 and no children aged 6 or over:

$$\begin{aligned} \frac{p}{1-p} &= 1.56 \times 0.42^1 \times 1.03^0 \\ &= 1.56 \times 0.42 \times 1 \\ &= 0.66 \end{aligned}$$

And two children aged 0 to 5, again with no children aged 6 or over:

$$\begin{aligned} \frac{p}{1-p} &= 1.56 \times 0.42^2 \times 1.03^0 \\ &= 1.56 \times 0.18 \times 1 \\ &= 0.28 \end{aligned}$$

11.4 Interpret using predicted probabilities

This probably my favorite way to actually get to grips with how a model works and R does all the hard work for us.

There are two ways to do it in R. The second is easier (and you would be forgiven for trying it first) but the first explains what's going on.

11.4.1 Using a customs predictions table

The recipe is as follows. First, generate a table with the values for predictors you would like to predict. This is a data frame with one column for all variables in the model.

The `expand.grid` command is handy for this as it generates all possible combinations of the values you provide.

```
predictions <- expand.grid(k5   = 0:4,
                           k618 = c(0,4,100))
predictions
```

k5 <int>	k618 <dbl>
0	0
1	0
2	0
3	0
4	0
0	4
1	4
2	4
3	4
4	4

1-10 of 15 rows Previous **1** 2 Next

Now use R's `predict` command. We can get the predictions on the log-odds scale or (more useful) as probabilities. For the latter, use the option `type = "response"`.

I generally save the result onto the predictions data frame:

```
predictions$lfp_logodds <- predict(mod_children, # the model
                                   newdata = predictions)

predictions$lfp_prob <- predict(mod_children,
                                newdata = predictions,
                                type = "response")
```

I will also add on the odds, for completeness:

```
predictions$lfp_odds <- exp(predictions$lfp_logodds)
```

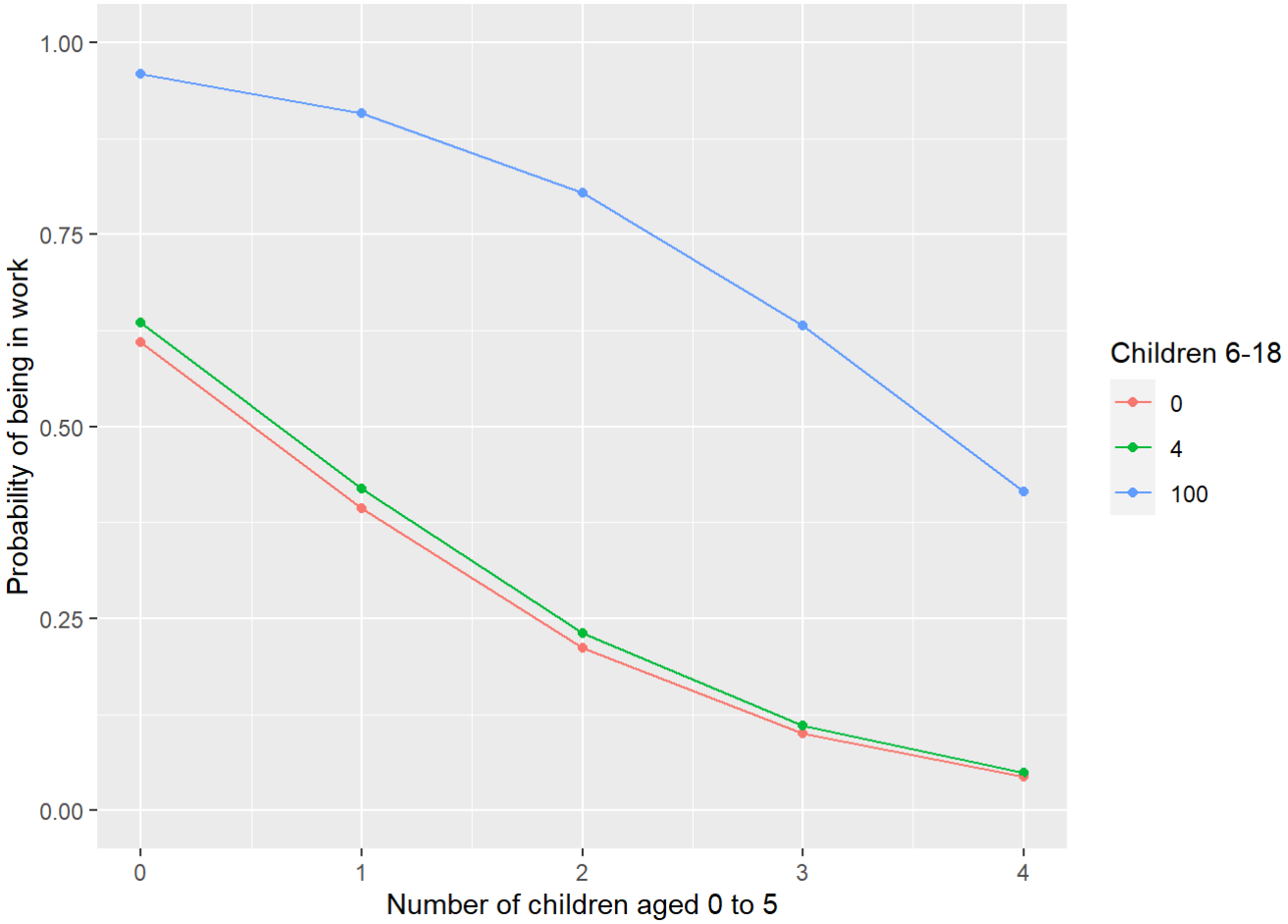
Have a look (I'm piping them through `kabel` from the `knitr` package to change the number of decimal places):

```
predictions  %>%
  kable(digits = c(0,0,2,2,2))
```

k5	k618	lfp_logodds	lfp_prob	lfp_odds
0	0	0.44	0.61	1.56
1	0	-0.43	0.39	0.65
2	0	-1.31	0.21	0.27
3	0	-2.19	0.10	0.11
4	0	-3.07	0.04	0.05
0	4	0.55	0.64	1.74
1	4	-0.33	0.42	0.72
2	4	-1.20	0.23	0.30
3	4	-2.08	0.11	0.12
4	4	-2.96	0.05	0.05
0	100	3.17	0.96	23.92
1	100	2.30	0.91	9.93
2	100	1.42	0.80	4.12
3	100	0.54	0.63	1.71
4	100	-0.34	0.42	0.71

The challenge is to pick sensible values for the predictors which illustrate (to you and readers) how the model works. You could then plot these using `ggplot`.

```
predictions %>%
  ggplot(aes(x = k5,
             y = lfp_prob,
             colour = factor(k618))) +
  geom_point() +
  geom_line() +
  labs(x = "Number of children aged 0 to 5",
       y = "Probability of being in work",
       colour = "Children 6-18") +
  ylim(0,1)
```



Above we see the effect of having 0, 1, 2, 3 or 4 children aged five or under, when the person has 0, 4 or 100 children aged 6-18. You would normally not include the blue line, but it's here for illustration purposes.

This process works for linear models fitted using `lm` too, and can be particularly handy for making sense of interactions or other non-linear effects.

11.4.2 Use ggeffects

Every few months someone creates a helpful R package for creating or understanding models. The `ggeffects` package is tear-jerkingly beautiful. See Lüdecke (2018) for more information. There are also helpful examples on the package webpage (https://strengejacke.github.io/ggeffects/).

```
library(ggeffects)
```

```
## Warning: package 'ggeffects' was built under R version 4.1.2
```

The command for making predictions is called `ggpredict` and wants a model and at least one variable name. It can generate predictions for up to four variables whilst holding the others at some constant.

Let's start with one, `k5`.

```
summary(mod_children)
```

```
##
## Call:
## glm(formula = lfp_yes ~ k5 + k618, family = binomial, data = dat)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4566  -1.3712   0.9634   0.9953   1.7617
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.44479    0.11106   4.005 6.20e-05 ***
## k5          -0.87926    0.15817  -5.559 2.71e-08 ***
## k618         0.02730    0.05767   0.473    0.636
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1029.75  on 752  degrees of freedom
## Residual deviance:  994.53  on 750  degrees of freedom
## AIC: 1000.5
##
## Number of Fisher Scoring iterations: 4
```

```
ggpredict(mod_children, terms = "k5")
```

x <int>	predicted <dbl>	std.error <dbl>	conf.low <dbl>	conf.high <dbl>	group <fct>
0	0.6158776	0.08429617	0.57612093	0.6541438	1
1	0.3995904	0.14817992	0.33234469	0.4708469	1
2	0.2164575	0.29469513	0.13423485	0.3298554	1
3	0.1028741	0.44918776	0.04538694	0.2166497	1

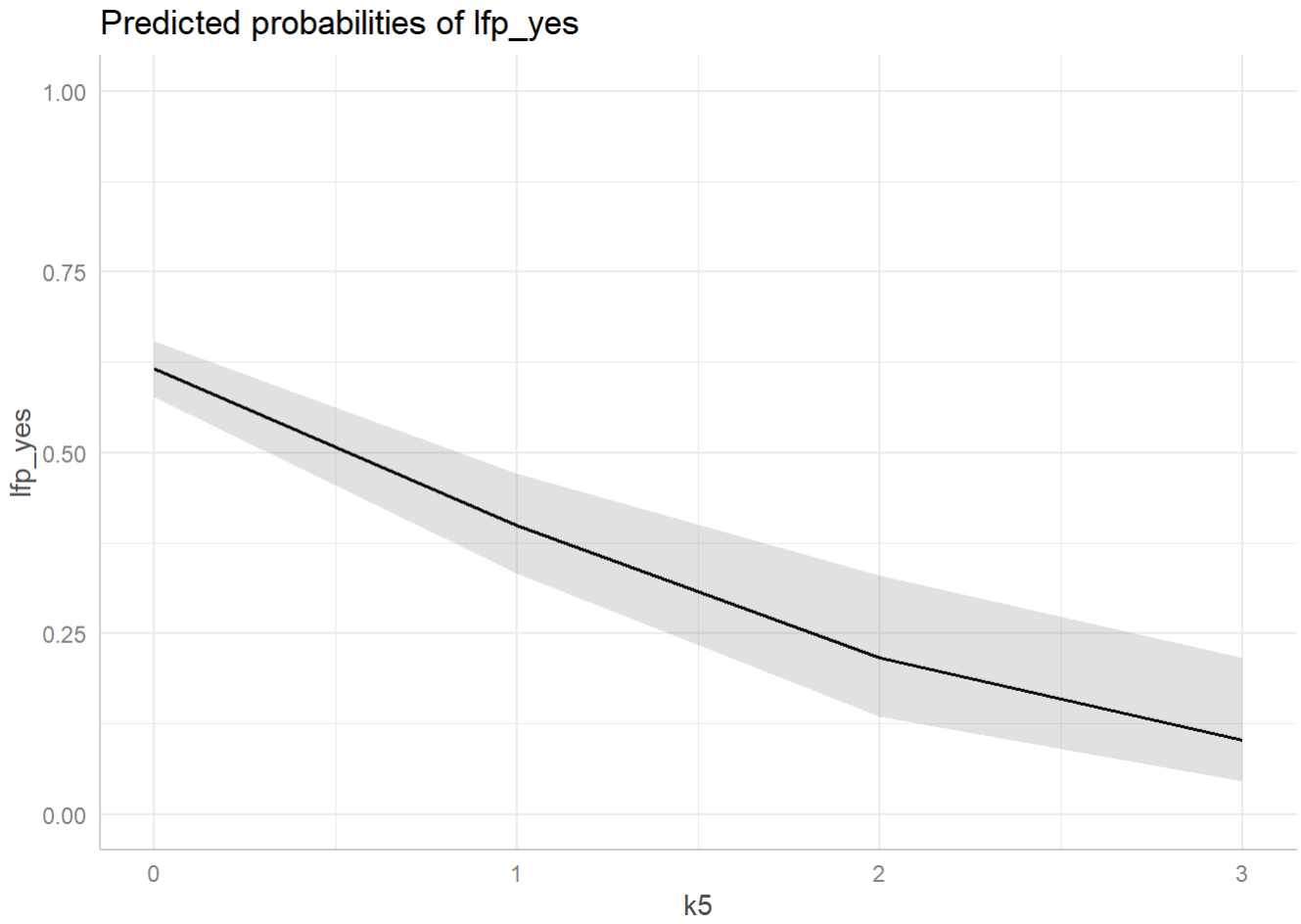
4 rows

It tries to guess sensible values to hold other predictions at. Here it has held `k618` at 1.

The package also comes with a `plot` function which automatically gives you a `ggplot` object.

```
ggpredict(mod_children, terms = c("k5")) %>%
  plot() +
  ylim(0,1)
```

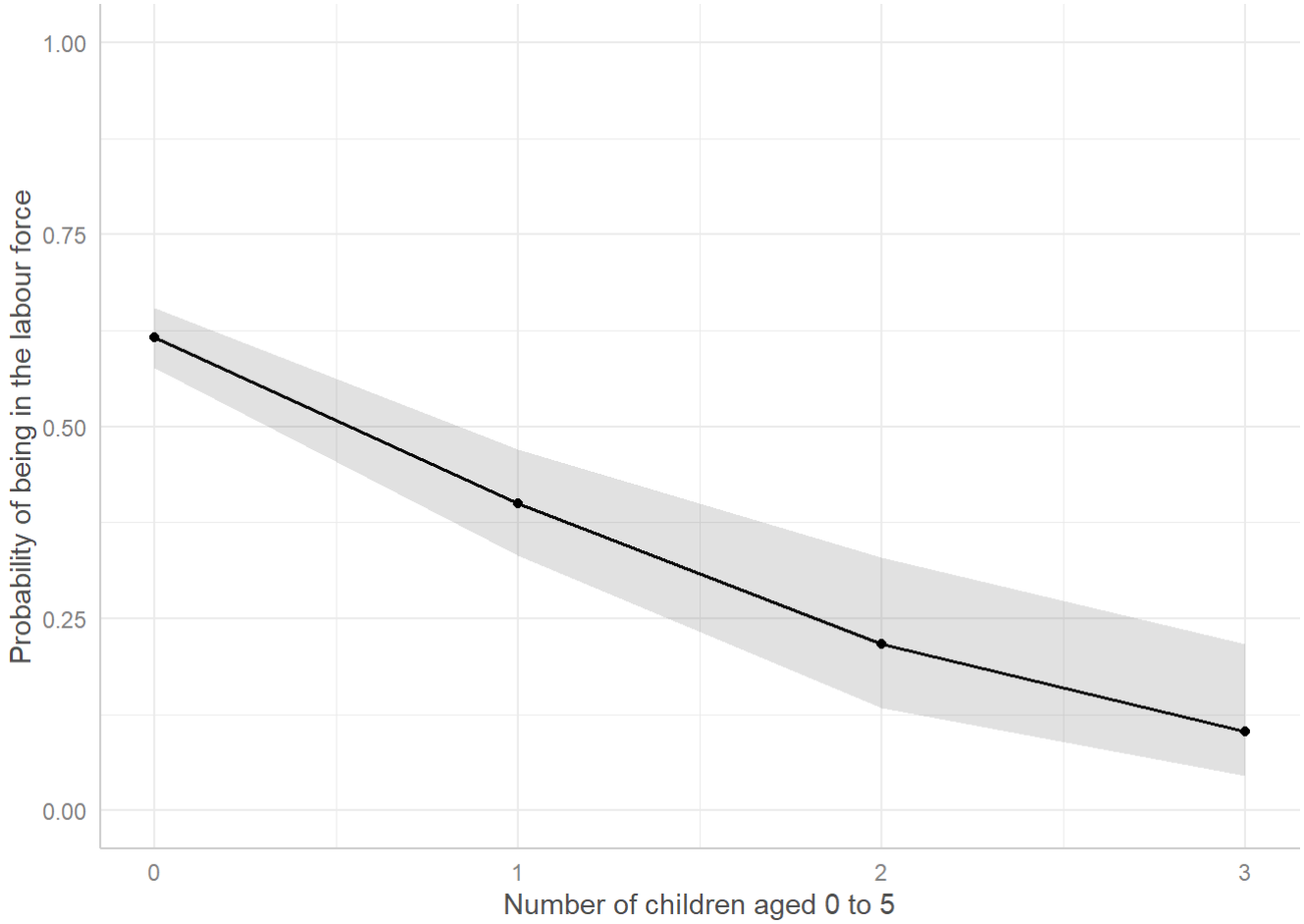
```
## Scale for 'y' is already present. Adding another scale for 'y', which
will
## replace the existing scale.
```



The grey area is a 95% confidence band.

You can then modify this plot as you would any other `ggplot` object:

```
ggpredict(mod_children, terms = c("k5")) %>%
  plot() +
  geom_point() +
  xlab("Number of children aged 0 to 5") +
  ylab("Probability of being in the labour force") +
  labs(title = NULL) +
  ylim(0,1)
```



Before publication (or module assignment), be sure to explain in your Figure captions what exactly a graph shows, i.e., in this case that the number of children ages 6 to 18 has been held at 1.

12 Diagnostics

Good news: we will use the same diagnostic tools as for linear regression. Actually (slightly) fewer will now apply.

Load the `car` package:

```
library(car)
```

12.1 Check the residual distribution

You can check for outlying residuals as before:

```
outlierTest(mod_children)
```

```
## No Studentized residuals with Bonferroni p < 0.05
## Largest |rstudent|:
##    rstudent unadjusted p-value Bonferroni p
## 92 1.778802           0.075272           NA
```

Do not check if the residuals are normally distributed – they will not be and that is fine for logistic regression models.

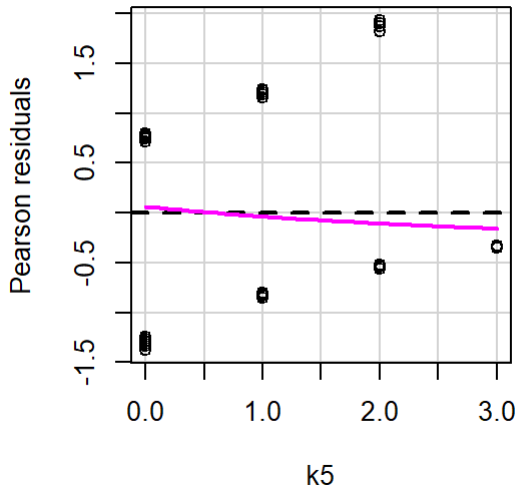
12.2 Check that the residual mean is constant

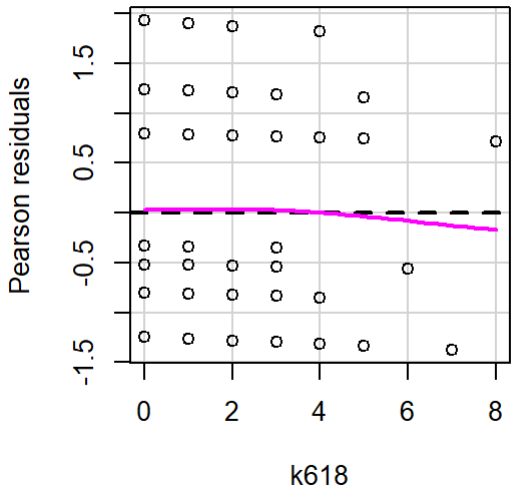
The `residualPlots` works as before, except when you run it with default setting on this model and data it will complain that "all data are on boundary of neighbourhood, make span bigger". The "span" refers to how much smoothing the smoother does!

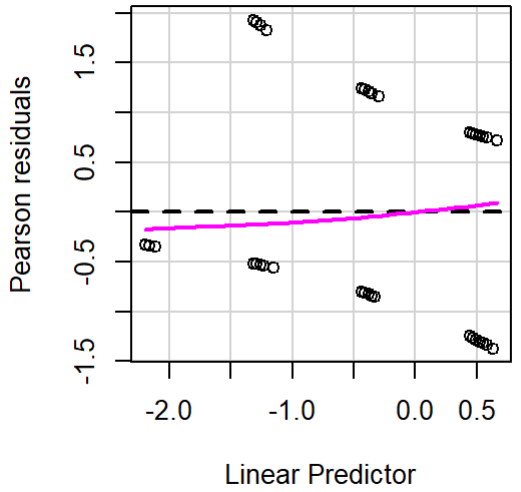
```
residualPlots(mod_children,
              tests = FALSE)
```

By default the span of the smoother is `2/3`. For this model and data, setting it to `1` works fine:

```
residualPlots(mod_children,
              tests = FALSE,
              smooth = list(span = 1))
```





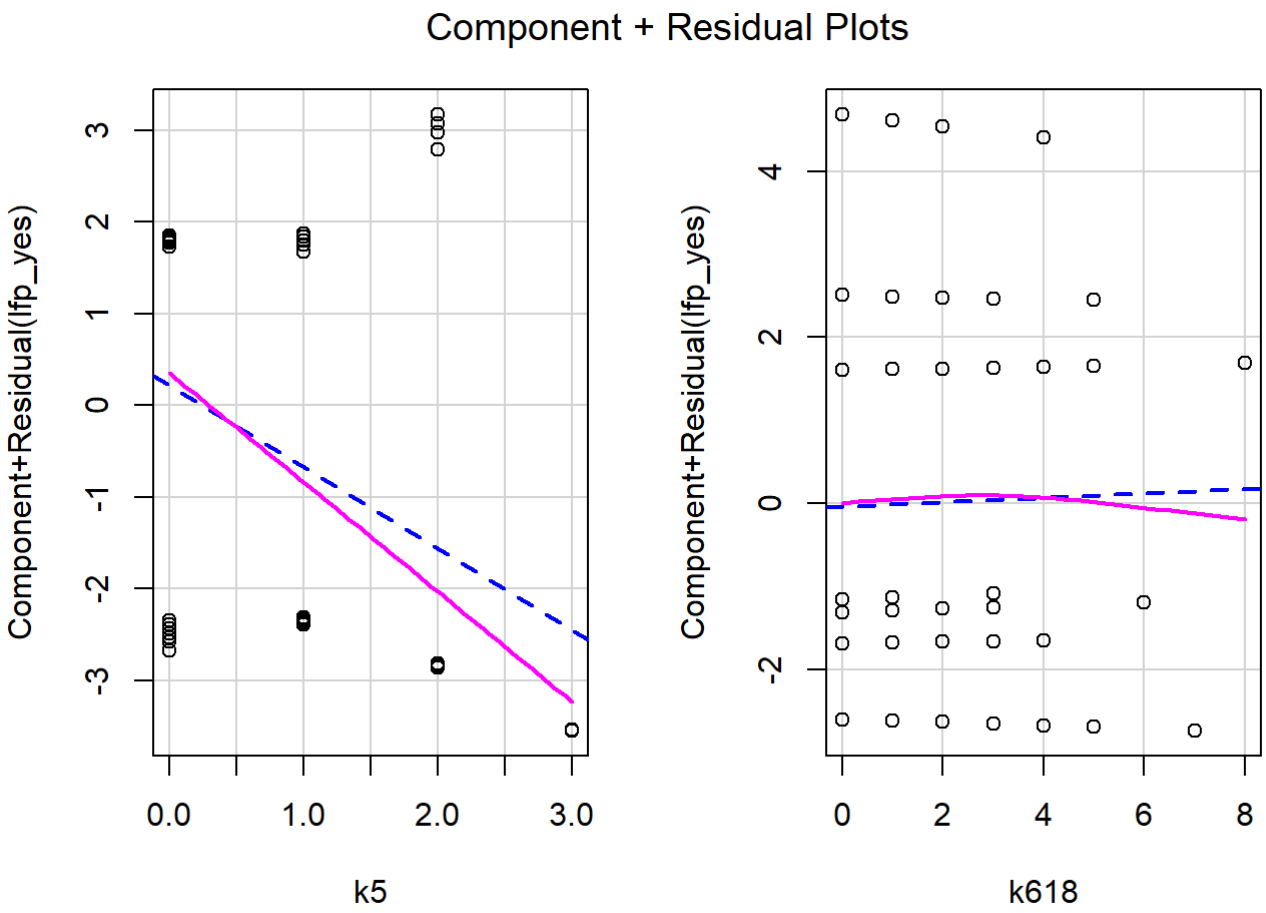


The funny patterns in the data points you see here are due to the outcome variable only taking on a 0 or a 1. This is to be expected. Focus on the magenta curves and ensure that they aren't curves (if you see what I mean...?): they should be straight lines along zero. And indeed here they are.

12.3 Linearity of predictors

We can use `crPlots` again to obtain component-plus-residual plots.

```
crPlots(mod_children, smooth = list(span = 1))
```



These too are fine – no obvious curves, though potentially something worth exploring towards the upper end of `k618` where there isn't much data (i.e., few families over 6 children). Again I had to fiddle with the `span` parameter to stop the smoother complaining.

12.4 Influence

Again everything we covered in the previous session applies here. Here's a quick way to look at potentially influential points:

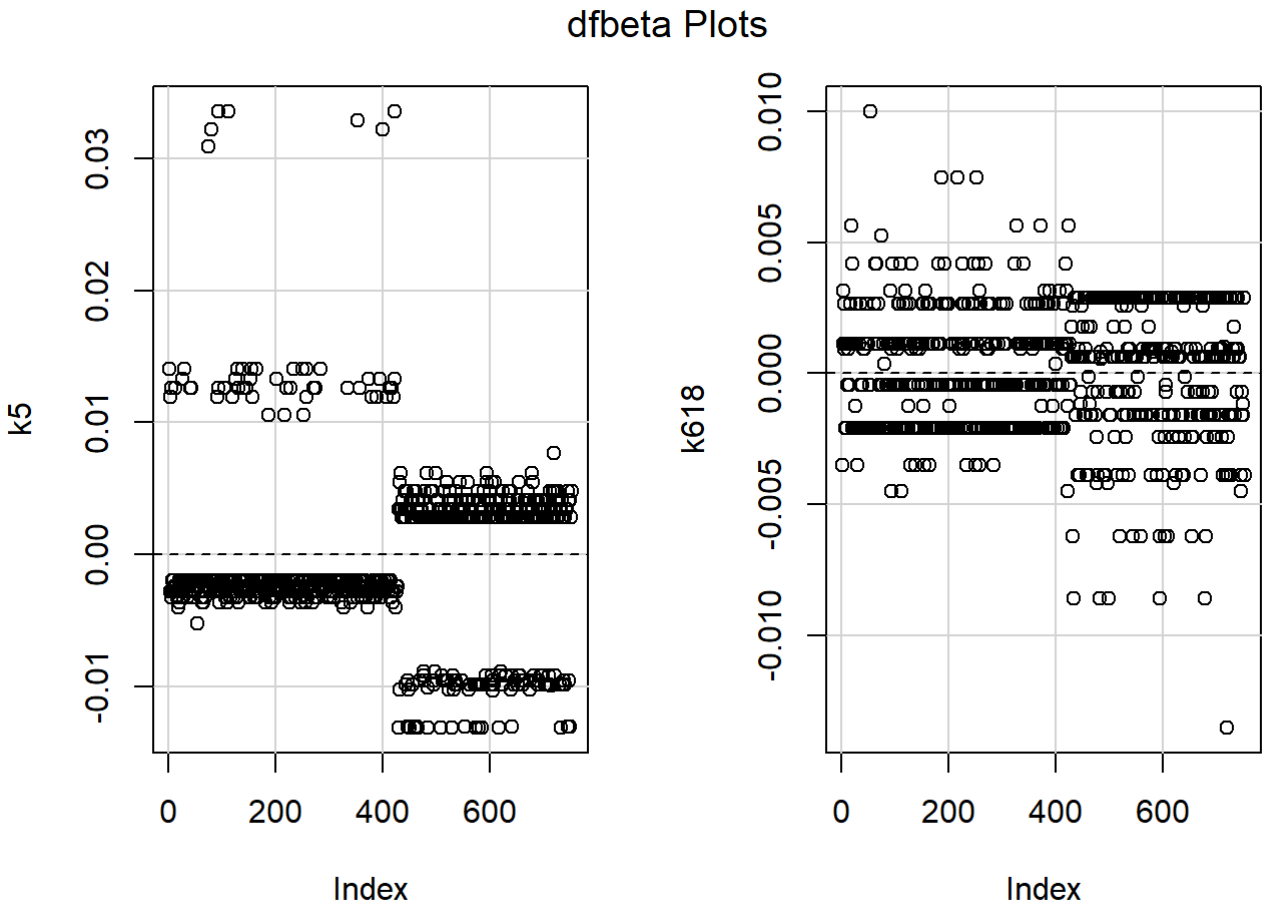
```
influence.measures(mod_children) %>%
  summary()
```

```
## Potentially influential observations of
##   glm(formula = lfp_yes ~ k5 + k618, family = binomial, data = dat) :
##
##     dfb.1_ dfb.k5 dfb.k618 dffit   cov.r   cook.d hat
## 18  -0.03  -0.02   0.09     0.09    1.01_*  0.00   0.01_*
## 53  -0.08  -0.03   0.15     0.15    1.04_*  0.01   0.04_*
## 74  -0.08   0.17   0.08     0.20_*  1.01_*  0.02   0.02_*
## 79  -0.03   0.18   0.01     0.18    1.01    0.02   0.01_*
## 92   0.02   0.18  -0.07     0.20_*  1.01    0.02   0.02_*
## 111  0.02   0.18  -0.07     0.20_*  1.01    0.02   0.02_*
## 186 -0.07   0.06   0.11     0.14    1.01_*  0.01   0.01_*
## 217 -0.07   0.06   0.11     0.14    1.01_*  0.01   0.01_*
## 252 -0.07   0.06   0.11     0.14    1.01_*  0.01   0.01_*
## 327 -0.03  -0.02   0.09     0.09    1.01_*  0.00   0.01_*
## 352  0.00   0.18  -0.03     0.19    1.01    0.02   0.01_*
## 371 -0.03  -0.02   0.09     0.09    1.01_*  0.00   0.01_*
## 400 -0.03   0.18   0.01     0.18    1.01    0.02   0.01_*
## 423  0.02   0.18  -0.07     0.20_*  1.01    0.02   0.02_*
## 424 -0.03  -0.02   0.09     0.09    1.01_*  0.00   0.01_*
## 430 -0.01  -0.07   0.03    -0.08    1.02_*  0.00   0.02_*
## 434  0.05   0.03  -0.13    -0.14    1.01    0.01   0.01_*
## 447  0.02  -0.07  -0.02    -0.08    1.02_*  0.00   0.02_*
## 450 -0.01  -0.07   0.03    -0.08    1.02_*  0.00   0.02_*
## 459 -0.01  -0.07   0.03    -0.08    1.02_*  0.00   0.02_*
## 462  0.01  -0.07   0.00    -0.08    1.02_*  0.00   0.01_*
## 463  0.02  -0.07  -0.02    -0.08    1.02_*  0.00   0.02_*
## 466 -0.01  -0.07   0.03    -0.08    1.02_*  0.00   0.02_*
## 482  0.05   0.03  -0.13    -0.14    1.01    0.01   0.01_*
## 483  0.00  -0.07   0.01    -0.07    1.02_*  0.00   0.01_*
## 484  0.00  -0.06   0.01    -0.06    1.02_*  0.00   0.02_*
## 499  0.05   0.03  -0.13    -0.14    1.01    0.01   0.01_*
## 509 -0.01  -0.07   0.03    -0.08    1.02_*  0.00   0.02_*
## 528 -0.01  -0.07   0.03    -0.08    1.02_*  0.00   0.02_*
## 553  0.01  -0.07   0.00    -0.08    1.02_*  0.00   0.01_*
## 574 -0.01  -0.07   0.03    -0.08    1.02_*  0.00   0.02_*
## 580  0.00  -0.07   0.01    -0.07    1.02_*  0.00   0.01_*
## 585  0.00  -0.07   0.01    -0.07    1.02_*  0.00   0.01_*
## 595  0.05   0.03  -0.13    -0.14    1.01    0.01   0.01_*
## 605  0.02  -0.06  -0.01    -0.06    1.02_*  0.00   0.02_*
## 617  0.00  -0.07   0.01    -0.07    1.02_*  0.00   0.01_*
## 641  0.01  -0.07   0.00    -0.08    1.02_*  0.00   0.01_*
## 678  0.05   0.03  -0.13    -0.14    1.01    0.01   0.01_*
## 715  0.00  -0.05   0.02    -0.06    1.02_*  0.00   0.02_*
## 720  0.10   0.04  -0.20    -0.21_*  1.02_*  0.02   0.03_*
## 732 -0.01  -0.07   0.03    -0.08    1.02_*  0.00   0.02_*
```

```
## 746   0.05   -0.07   -0.07     -0.11     1.03_*   0.00    0.03_*
## 750   0.02   -0.07   -0.02     -0.08     1.02_*   0.00    0.02_*
```

We could also look at the DFBETA plots:

```
dfbetaPlots(mod_children)
```



The numbers are minuscule, relative to the model slopes, so nothing to worry about. I am curious about the largest DFBETA for `k618`, though. The raw descriptives showed one person with eight children aged 6 to 18 who was in work. Is the slope being ever so slightly dragged up by her data?

```
dfbeta(mod_children) %>%
  as.data.frame() %>%
  filter(k618 > .009)
```

	(Intercept) <dbl>	k5 <dbl>	k618 <dbl>
53	-0.009795223	-0.005237558	0.009994706

1 row

That's row 53, so let's `slice` the data to look:

```
dat %>%
  slice(53)
```

lfp	lfp_yes	k5	k618	age	wc	hc	lwg	inc
<chr>	<int>	<int>	<int>	<int>	<fct>	<fct>	<dbl>	<dbl>
yes	1	0	8	37	no	no	0.1625193	16.258

1 row

Yes, this is the one person with 8 children! But it doesn't matter since the dataset is so large and the DFBETA minuscule.

12.5 Multicollinearity

Yep, you can still check the variance inflation factors (VIFs) and interpret as before!

```
vif(mod_children)
```

```
##       k5     k618
## 1.010858 1.010858
```

These are both close to 1 so all is good.

As a conceptual exercise, we could try adding multicollinearity to the model, just to be sure that the VIFs work... Here I'm adding a new variable called `kids` which is the sum of `k5` and `k618`:

```
dat <- dat %>%
  mutate(kids = k5 + k618)
```

Here's a model with both `k618` and `kids` as predictors:

```
high_vif_maybe <- glm(lfp_yes ~ k618 + kids, data = dat, family = binomi
al)
```

And here are the VIFs:

```
vif(high_vif_maybe)
```

```
##     k618     kids
## 9.189766 9.189766
```

They are high as you might expect, given how the `kids` variable was created.

13 A challenge

13.1 Activity | 13.2 (An) Answer

There are several other variables in the dataset which you can now explore.

a. Does a model with the following predictors added predict better than one with only the number of children?

Variable name	Description
`age`	in years.
`wc`	wife's college attendance (no/yes).
`hc`	husband's college attendance (no/yes).
`inc`	family income exclusive of wife's income (in $1000s).

b. Interpret the coefficients, using a method of your choice
c. Try some diagnostics (this week you have some more creative freedom) – do you want to do anything as a result of what you find?

14 References

Fox, J. & Weisberg, S. (2019). *An R Companion to Applied Regression*, Third Edition, Sage.

Gelman, A., & Hill, J. (2007). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge: Cambridge University Press.

Lüdecke D (2018). ggeffects: Tidy Data Frames of Marginal Effects from Regression Models (https://joss.theoj.org/papers/10.21105/joss.00772). Journal of Open Source Software, 3(26), 772. doi: 10.21105/joss.00772

Mroz, T. A. (1987). The sensitivity of an empirical model of married women's hours of work to economic and statistical assumptions. *Econometrica*, *55*, 765–799.