Binomial Generalized Linear Models

Filippo Gambarota

University of Padova

2022/2023

Updated on 2023-04-27

Outline

Binomial GLM

Example: Passing the exam

We want to measure the impact of **watching tv-shows** on the probability of **passing the statistics exam**.

- `exam`: **passing the exam** (1 = "passed", 0 = "failed")
- `tv_shows`: **watching tv-shows regularly** (1 = "yes", 0 = "no")

```
head(dat)
```

```
##   tv_shows exam
## 1        1    0
## 2        1    1
## 3        1    1
## 4        1    1
## 5        1    1
## 6        1    0
```

Example: Passing the exam

We can create the **contingency table**

```
xtabs(~exam + tv_shows, data = dat) |>
    addmargins()
```

```
##      tv_shows
## exam    0   1 Sum
##    0   31  22  53
##    1   19  28  47
##    Sum 50  50 100
```

Example: Passing the exam

Each cell probability π_{ij} is computed as π_{ij}/n

```
(xtabs(~exam + tv_shows, data = dat)/n) |>
    addmargins()
```

```
##      tv_shows
## exam     0    1  Sum
##   0   0.31 0.22 0.53
##   1   0.19 0.28 0.47
##   Sum 0.50 0.50 1.00
```

Example: Passing the exam - Odds

The most common way to analyze a 2x2 contingency table is using the **odds ratio** (OR). Firsly let's define *the odds of success* as:

$$odds = \frac{\pi}{1-\pi}$$
$$\pi = \frac{odds}{odds+1}$$

- the **odds** are non-negative, ranging between 0 and $+\infty$
- an **odds** of e.g. 3 means that we expect 3 *success* for each *failure*

Example: Passing the exam - Odds

For the `exam` example:

```
odds <- function(p) p / (1 - p)
p11 <- mean(with(dat, exam[tv_shows == 1])) # passing exam | tv_shows
odds(p11)
```

```
## [1] 1.272727
```

Example: Passing the exam - Odds Ratio

The OR is a ratio of odds:

$$OR = \frac{\frac{\pi_1}{1-\pi_1}}{\frac{\pi_2}{1-\pi_2}}$$

- OR ranges between 0 and $+\infty$. When $OR = 1$ the odds for the two conditions are equal
- An e.g. $OR = 3$ means that being in the condition at the numerator increase 3 times the odds of success

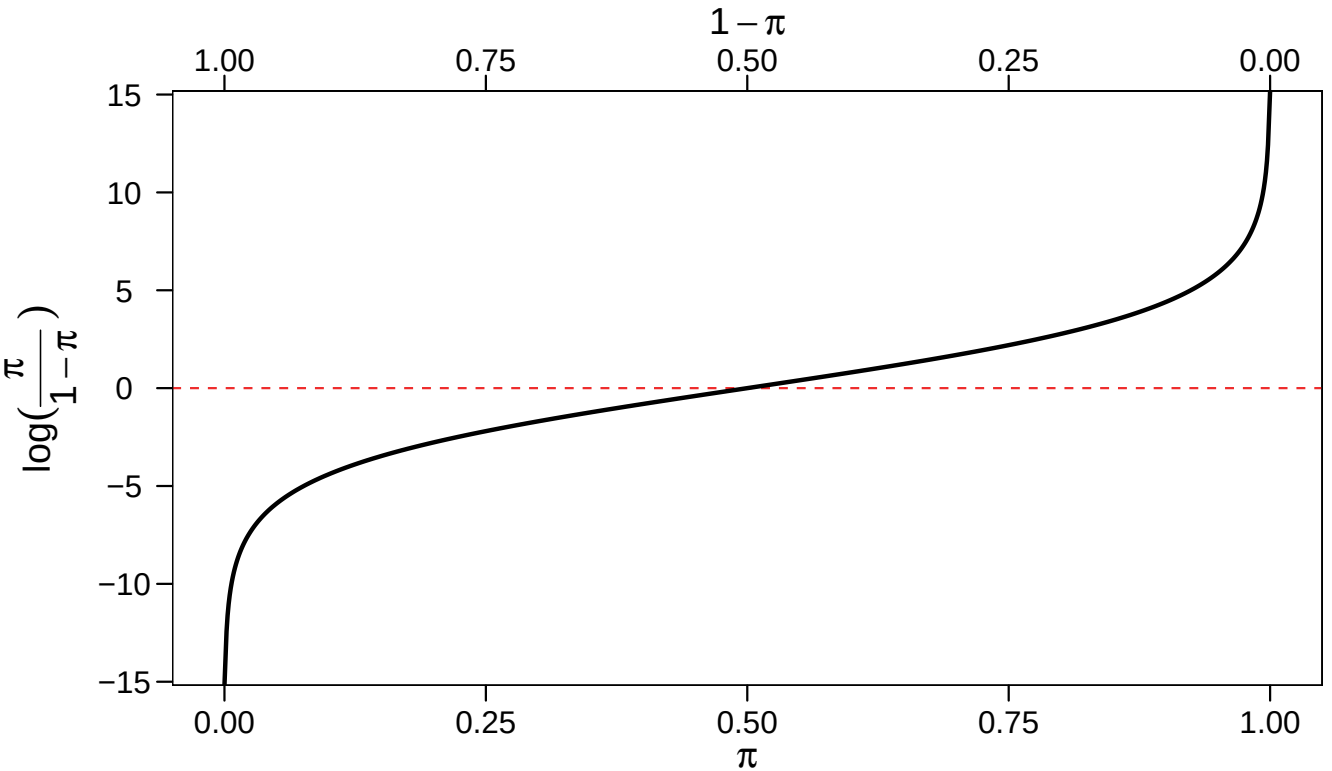
Example: Passing the exam - Odds Ratio

```
odds_ratio <- function(p1, p2) odds(p1) / odds(p2)
p11 <- mean(with(dat, exam[tv_shows == 1])) # passing exam | tv_shows
p10 <- mean(with(dat, exam[tv_shows == 0])) # passing exam | not tv_shows
odds_ratio(p11, p10)
```

```
## [1] 2.076555
```

Why using these measure?

The odds have an interesting property when taking the logarithm. We can express a probability π using a scale ranging $[-\infty, +\infty]$



Another example, **Teddy Child**

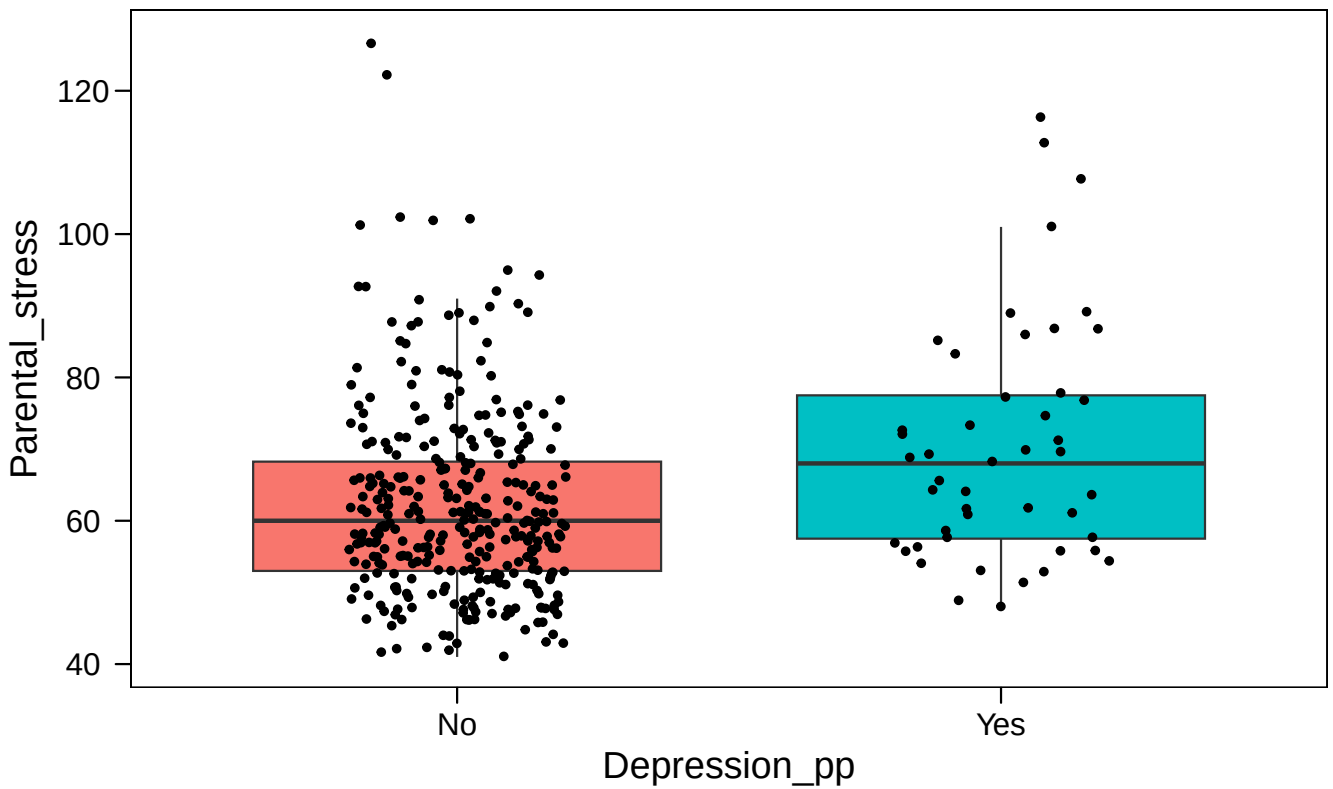
We considered a Study conducted by the University of Padua (TEDDY Child Study, 2020)[1]. Within the study, researchers asked the participants (mothers of a young child) about the presence of post-partum depression and measured the parental stress using the PSI-Parenting Stress Index.

ID	Parental.stress	Depression.pp
1	75	No
2	51	No
3	76	No
4	88	No
...	...	...
376	67	No
377	71	No
378	63	No
379	70	No

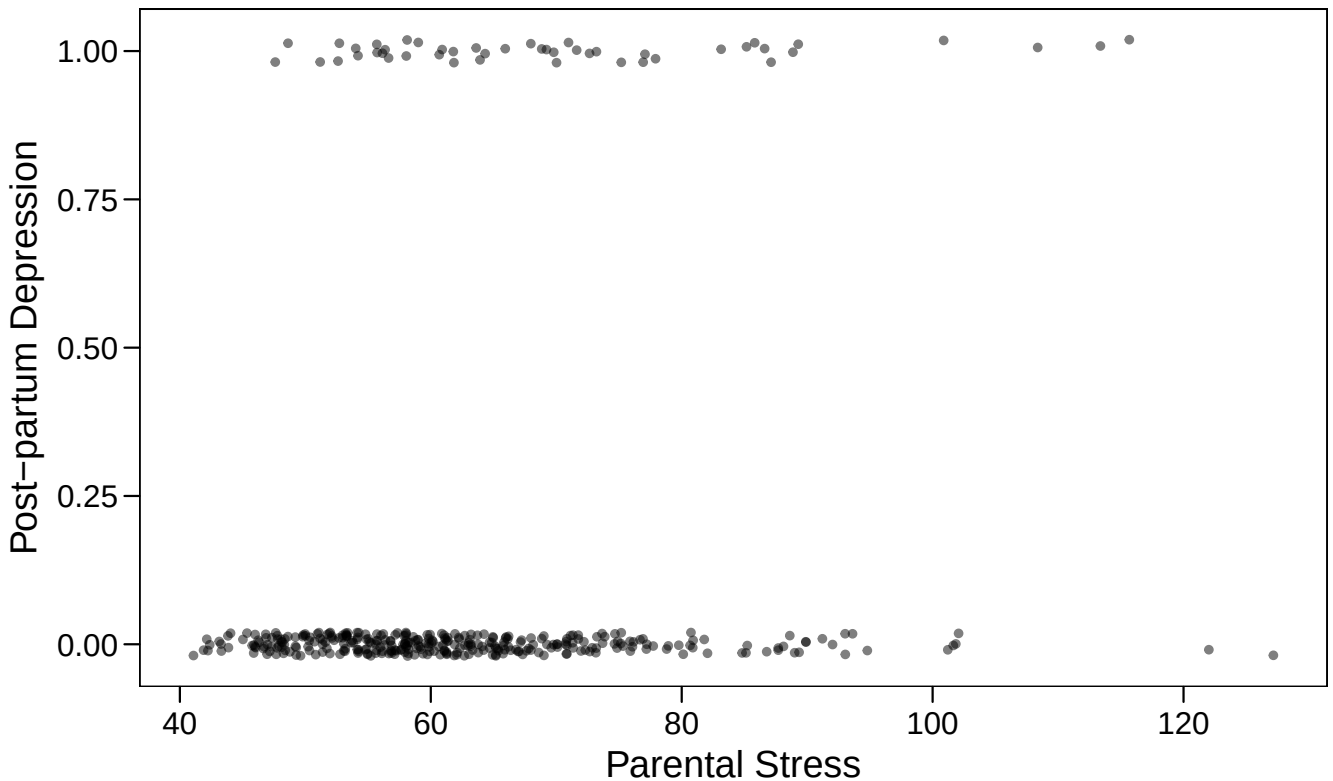
[1] Thanks to Prof. Paolo Girardi for the example, see https://teddychild.dpss.psy.unipd.it/ for information

Another example, **Teddy Child**

We want to see if the parental stress increase the probability of having post-partum depression:



Another example, **Teddy Child**



Another example, **Teddy Child**

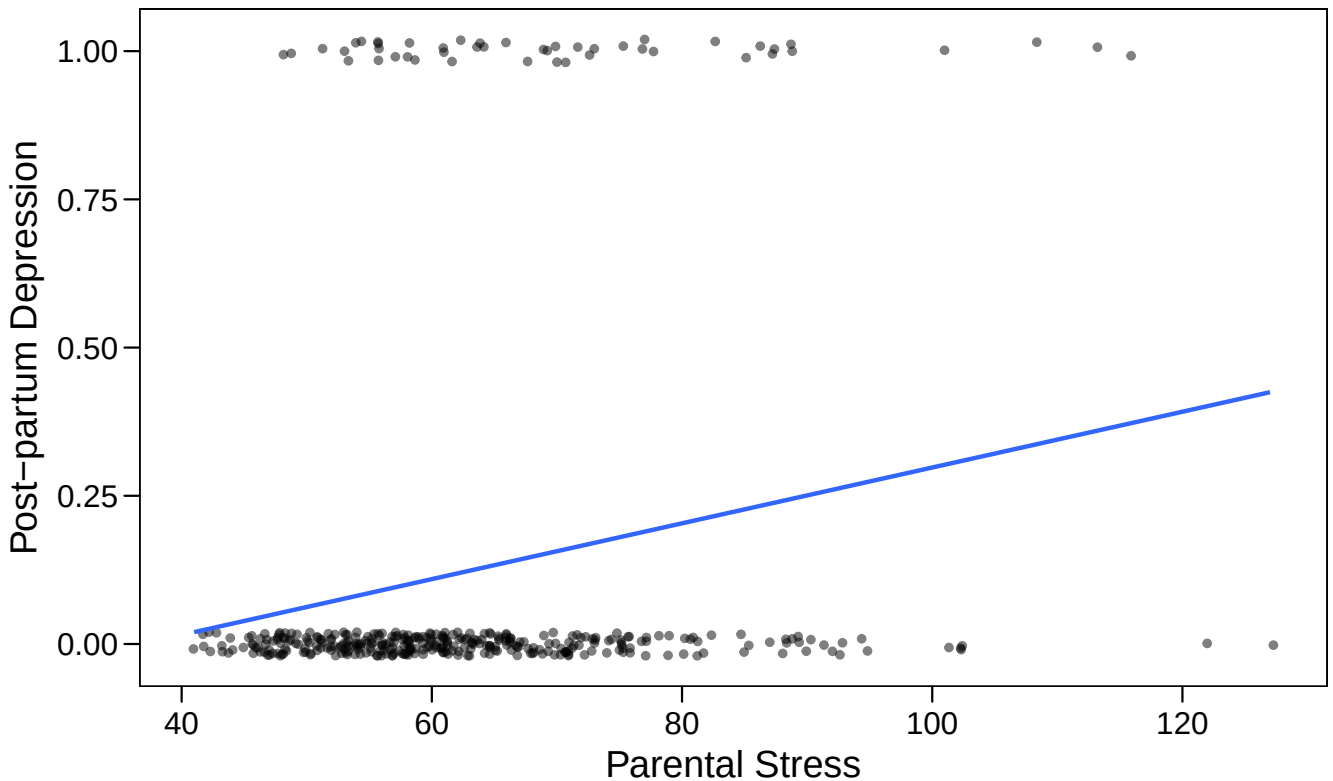
Let's start by fitting a linear model `Depression_pp ~ Parental_stress`. We consider "Yes" as 1 and "No" as 0.

```
fit_lm <- lm(Depression_pp01 ~ Parental_stress, data = teddy)
summary(fit_lm)
```

```
##
## Call:
## lm(formula = Depression_pp01 ~ Parental_stress, data = teddy)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.42473 -0.13768 -0.10003 -0.05768  0.94702
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -0.172900   0.077561  -2.229 0.026389 *
## Parental_stress  0.004706   0.001201   3.919 0.000105 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3239 on 377 degrees of freedom
## Multiple R-squared:  0.03915,    Adjusted R-squared:  0.0366
## F-statistic: 15.36 on 1 and 377 DF,  p-value: 0.0001054
```

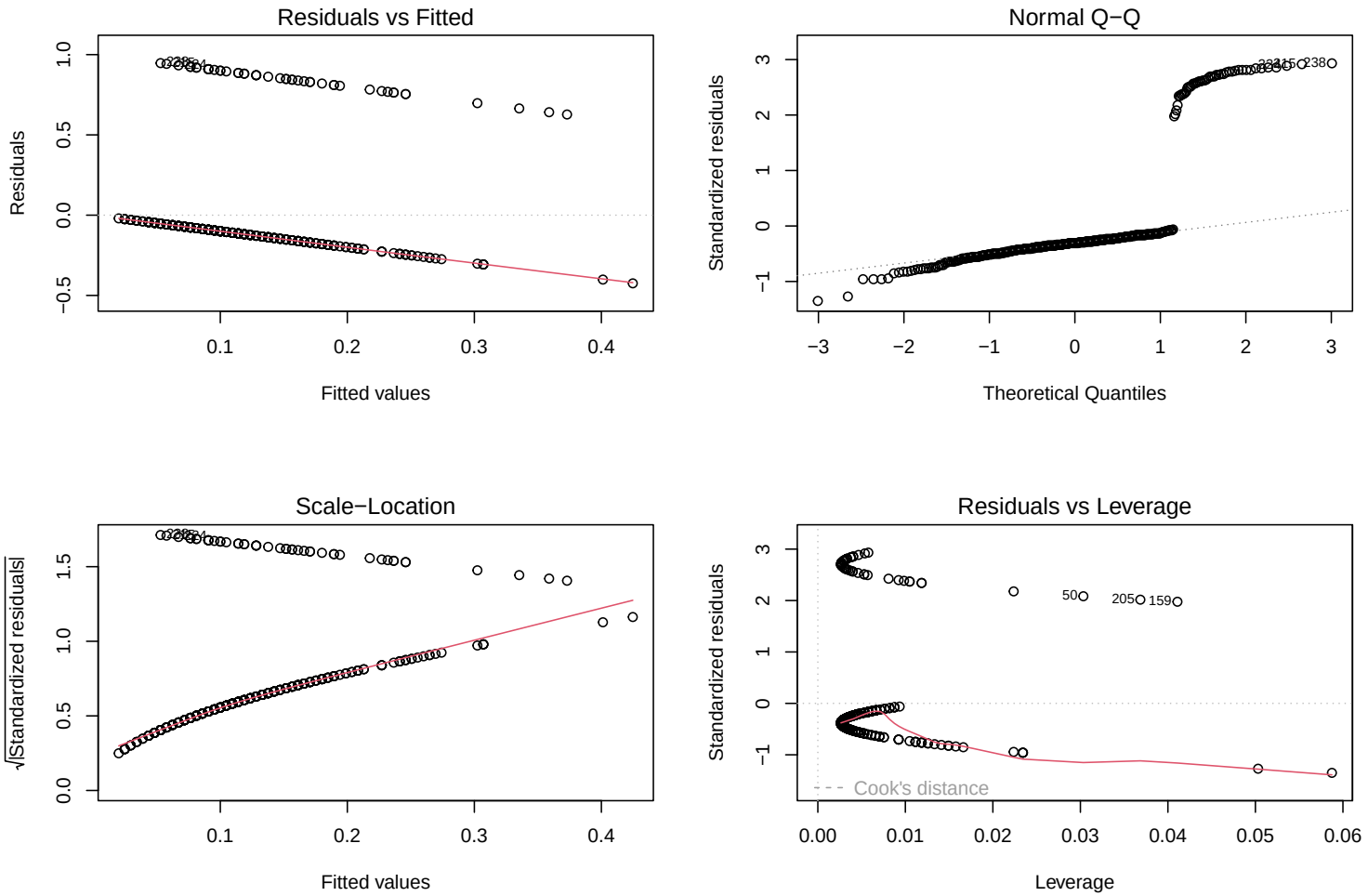
Another example, **Teddy Child**

Let's add the fitted line to our plot:



Another example, **Teddy Child**

... and check the residuals, pretty bad right?



Another example, **Teddy Child**

As for the exam example, we could compute a sort of contingency table despite the `Parental_stress` is a numerical variable by creating some discrete categories (just for exploratory analysis):

```
table(teddy$Depression_pp, teddy$Parental_stress_c) |>
    round(2)
```

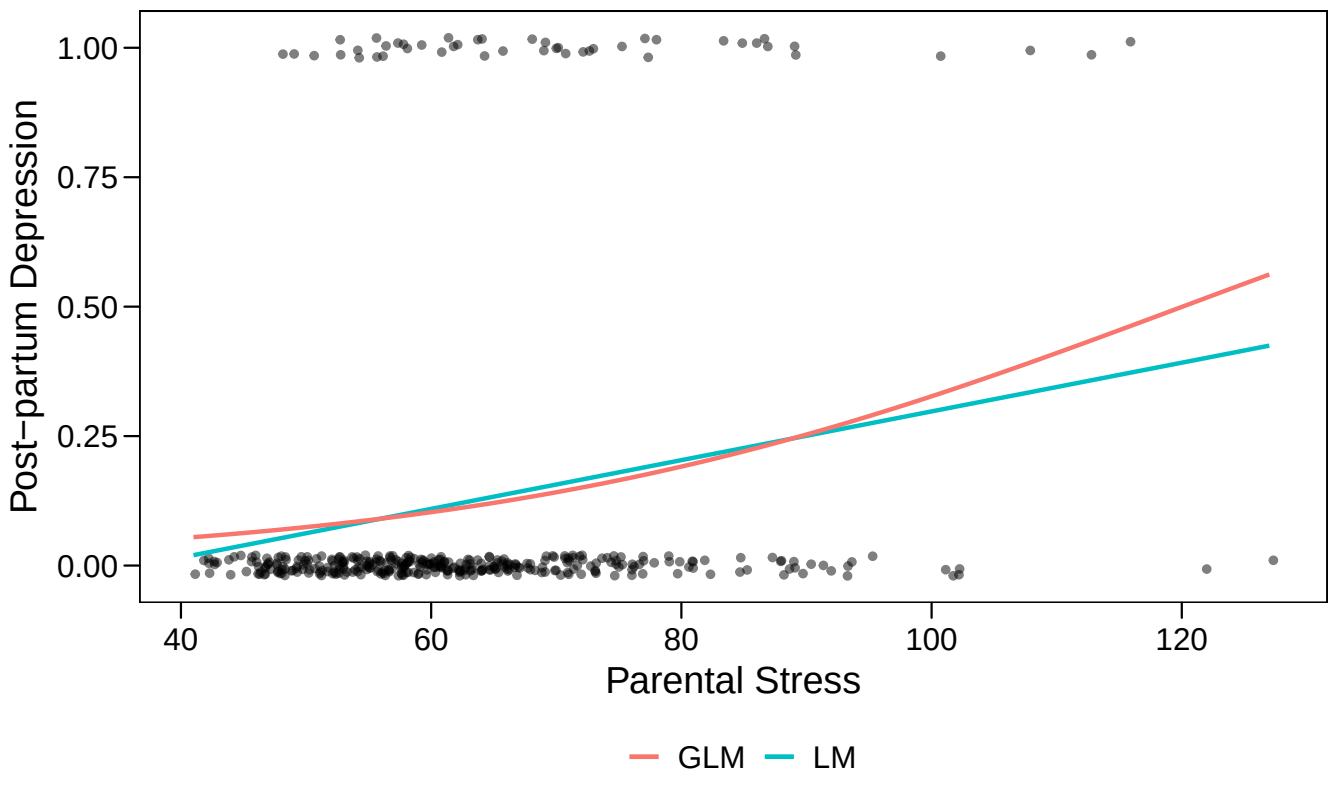
```
##
##       < 40 40-60 60-80 80-100 > 100
##   No     0   164   136      26     6
##   Yes    0    15    21       7     4
```

```
table(teddy$Depression_pp, teddy$Parental_stress_c) |>
    prop.table(margin = 2) |>
    round(2)
```

```
##
##       < 40 40-60 60-80 80-100 > 100
##   No   0.92  0.87  0.79  0.60
##   Yes  0.08  0.13  0.21  0.40
```

Another example, **Teddy Child**

Ideally, we could compute the increase in the odds of having the post-partum depression as the parental stress increase. In fact, as we are going to see, the Binomial GLM is able to estimate the non-linear increase in the probability.



Binomial GLM

- The **random component** of a Binomial GLM the binomial distribution with parameter π
- The **systematic component** is a linear combination of predictors and coefficients βX
- The **link function** is a function that map probabilities into the $[-\infty, +\infty]$ range.

Binomial GLM - Logit Link

The **logit** link is the most common link function when using a binomial GLM:

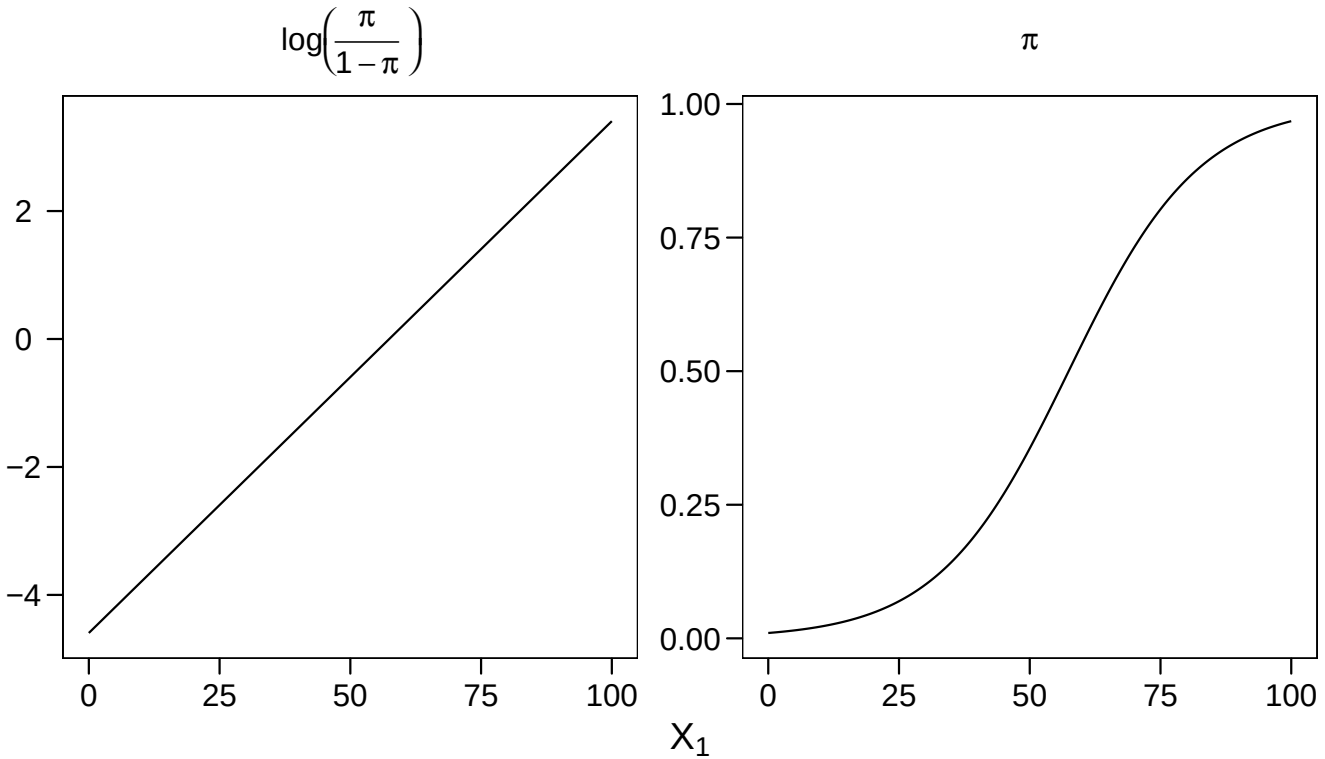
$$log\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 X_1 + ...\beta_p X_p$$

The inverse of the **logit** maps again the probability into the $[0, 1]$ range:

$$\pi = \frac{e^{\beta_0+\beta_1 X_1+...\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+...\beta_p X_p}}$$

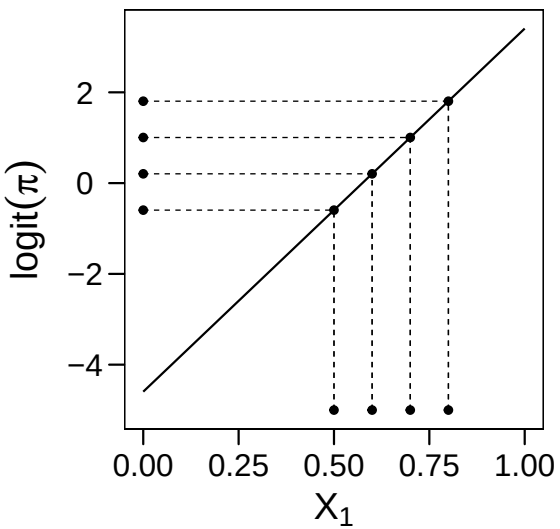
Binomial GLM - Logit Link

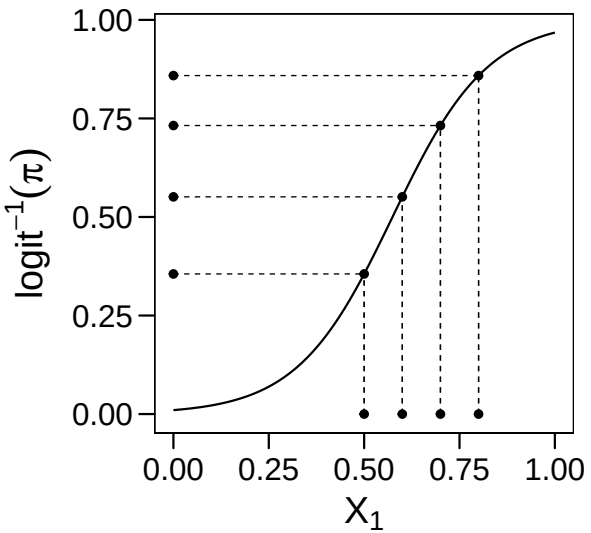
Thus with a single numerical predictor x the relationship between x and π in non-linear on the probability scale but linear on the logit scale.



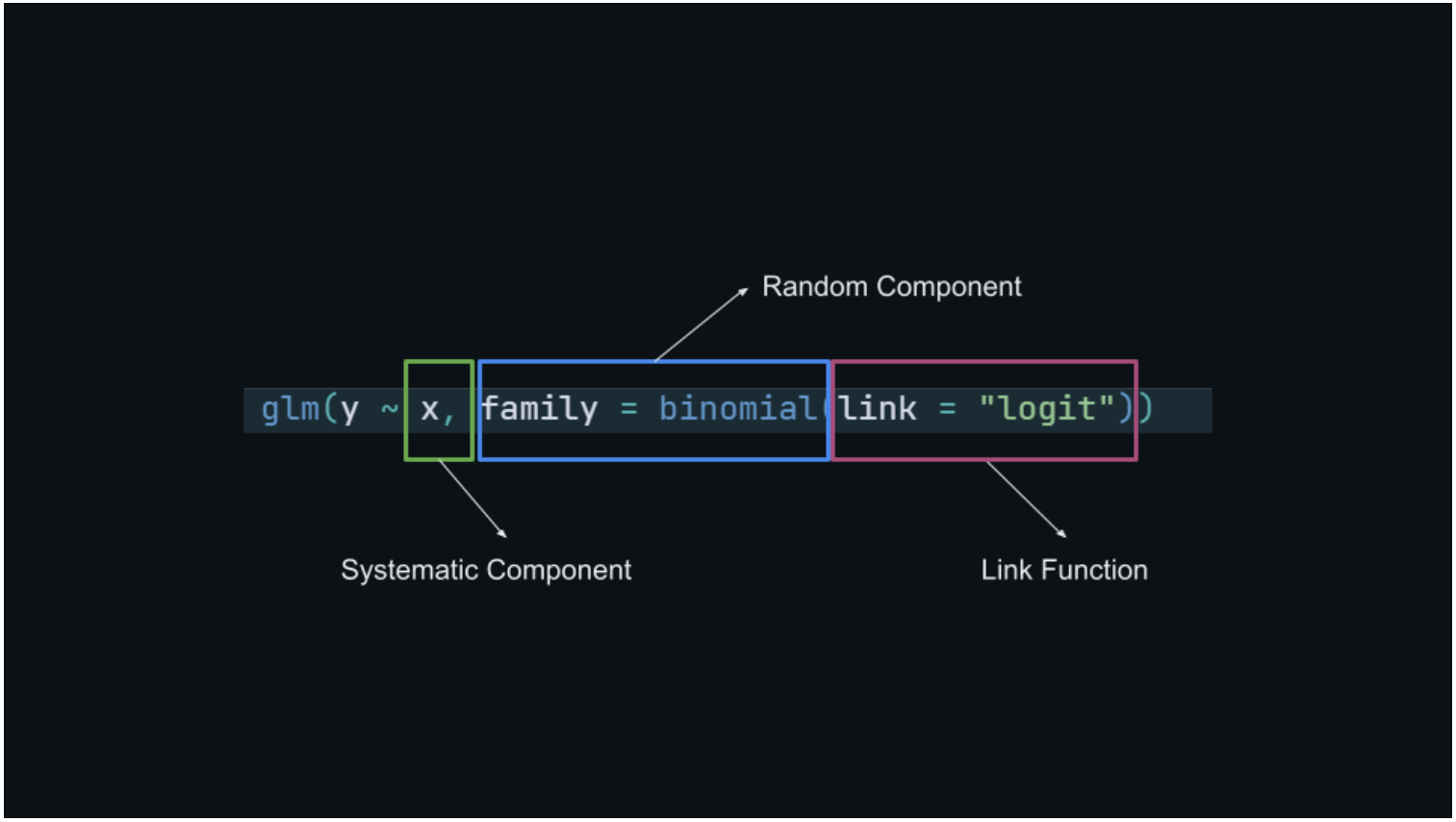
Binomial GLM - Logit Link

The problem is that effects are non-linear, thus is more difficult to interpret and report model results

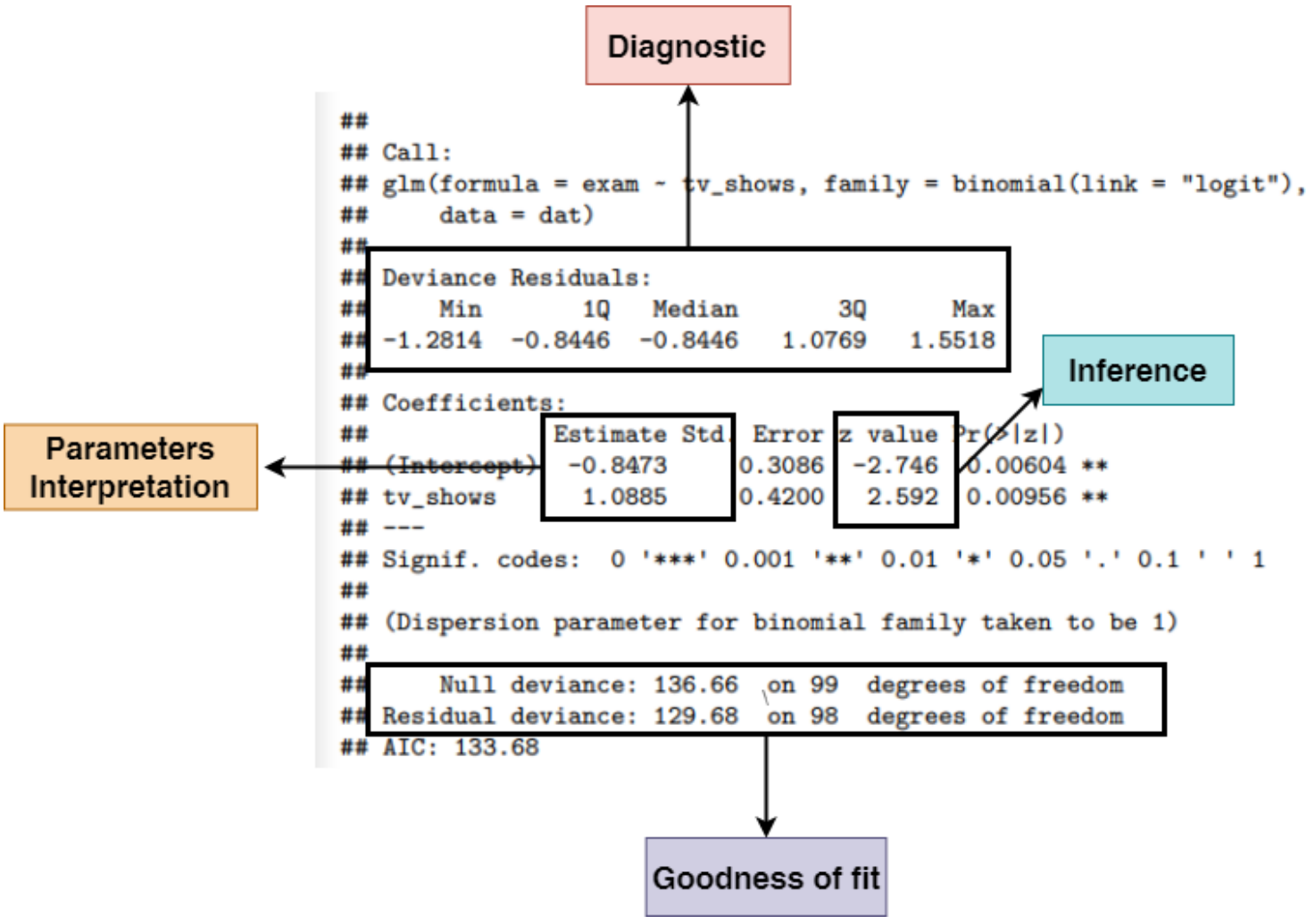




Binomial GLM - Model fitting in R



The big picture...



Binomial GLM - Model fitting in R

We can model the contingency table presented before. We put data in **binary form**:

```
##     tv_shows
## exam  0  1
##    0 35 22
##    1 15 28
```

```
##   tv_shows exam
## 1        1    0
## 2        1    1
## 3        1    1
## 4        1    1
## 5      ...  ...
## 6        0    0
## 7        0    1
## 8        0    0
## 9        0    0
```

Binomial GLM - Intercept only model

Let's start from the simplest model (often called null model) where there are no predictors:

```
fit0 <- glm(exam ~ 1, data = dat, family = binomial(link = "logit"))
summary(fit0)
```

```
##
## Call:
## glm(formula = exam ~ 1, family = binomial(link = "logit"), data = dat)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.060  -1.060  -1.060   1.299   1.299
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.2819     0.2020  -1.395    0.163
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 136.66  on 99  degrees of freedom
## Residual deviance: 136.66  on 99  degrees of freedom
## AIC: 138.66
##
## Number of Fisher Scoring iterations: 4
```

Binomial GLM - Intercept only model

When fitting an intercept-only model, the parameter is the average value of the y variable:

$$\log(\frac{\pi}{1-\pi}) = \beta_0$$

$$\pi = \frac{e^{\beta_0}}{1+e^{\beta_0}}$$

Binomial GLM - Intercept only model

In R, the $logit(\pi)$ is computed using `qlogis()` that is the `q` + `logis` combination of functions to work with probability distributions. The $logit^{-1}$ thus the inverse of the logit function is `plogis()`:

```
# average y on the respo nse scale
mean(dat$exam)
```

```
## [1] 0.43
```

```
c("logit" = coef(fit0)[1],
  "inv-logit" = plogis(coef(fit0)[1])
)
```

```
##     logit.(Intercept) inv-logit.(Intercept)
##             -0.2818512              0.4300000
```

Binomial GLM - Link function (TIPS)

If you are not sure about how to transform using the link function you can directly access the `family()` object in R that contains the appropriate link function and the corresponding inverse.

```
bin <- binomial(link = "logit")
bin$linkfun() # the same as plogis
bin$linkinv() # the same as qlogis
```

Binomial GLM - Model with X

Now we can add the `tv_shows` predictor. Now the model has two coefficients. Given that the `tv_shows` is a binary variable, the intercept is the average `y` when `tv_shows` is 0, and the `tv_shows` coefficient is the increase in `y` for a unit increase in `tv_shows`:

```
fit <- glm(exam ~ tv_shows, data = dat, family = binomial(link = "logit"))
summary(fit)
```

```
##
## Call:
## glm(formula = exam ~ tv_shows, family = binomial(link = "logit"),
##     data = dat)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2814  -0.8446  -0.8446   1.0769   1.5518
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.8473     0.3086  -2.746  0.00604 **
## tv_shows      1.0885     0.4200   2.592  0.00956 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 136.66  on 99  degrees of freedom
## Residual deviance: 129.68  on 98  degrees of freedom
## AIC: 133.68
```

Binomial GLM - Model with X

Thinking about our data, the `(Intercept)` is the probability of passing the exam without watching tv-shows. The `tv_shows` (should be) the difference in the probability of passing the exam between people who watched or did not watched tv-shows, BUT:

- we are on the logit scale. Thus we are modelling **log(odds)** and not probabilities
- a difference on the **log** scale is a ratio on the raw scale. Thus taking the exponential of `tv_shows` we obtain the ratio of odds of passing the exam watching vs non-watching tv-shows. Do you remember something?

Binomial GLM - Model with X_1

The `tv_shows` is exactly the Odds Ratio that we calculated on the contingency table:

```
# from model coefficients
exp(coef(fit)["tv_shows"])
```

```
## tv_shows
## 2.969697
```

```
# from the contingency table
odds_ratio(mean(dat$exam[dat$tv_shows == 1]),
           mean(dat$exam[dat$tv_shows == 0]))
```

```
## [1] 2.969697
```

Binomial GLM - Parameter Intepretation

Binomial GLM - Model Intepretation

Given the non-linearity and the link function, parameter intepretation is not easy for GLMs. In the case of the Binomial GLM we will se:

- interpreting model coefficients on the linear and logit scale
- odds ratio (already introduced)
- the divide by 4 rule [1], [2]
- marginal effects
- predicted probabilities

Binomial GLM - Intepreting model coefficients

Models coeffiecients are intepreted in the same way as standard regression models. The big difference concerns:

- numerical predictors
- categorical predictors

Using contrast coding and centering/standardizing we can make model coeffiecients more intepretable or tailored to our research question.

Binomial GLM - Categorical predictors

We we use a categorical predictor with p levels, the model will estimate $p-1$ parameters. The interpretation of these parameters is controlled by the contrast coding. In R the default is the treatment coding (or dummy coding). Essentially R create $p-1$ dummy variables (0 and 1) where 0 is the reference level (usually the first category) and 1 is the current level. We can see the coding scheme using the `model.matrix()` function that return the X matrix:

```
##   X.Intercept. tv_shows
## 1            1        1
## 2            1        1
## 3            1        1
## 4            1        1
## 5          ...      ...
## 6            1        0
## 7            1        0
## 8            1        0
## 9            1        0
```

Binomial GLM - Categorical predictors

In the simple case of the `exam` dataset, the intercept (β_0) is the reference level (default to 0 because is the first) and β_0 is the difference between the actual level and the reference level. If we change the order of the levels we could change the intercept value while β_1 will be the same. As an example we could use the so-called sum to zero coding where instead of assigning 0 and 1 we use different values. For example assigning -0.5 and 0.5 will make the intercept the grand-mean:

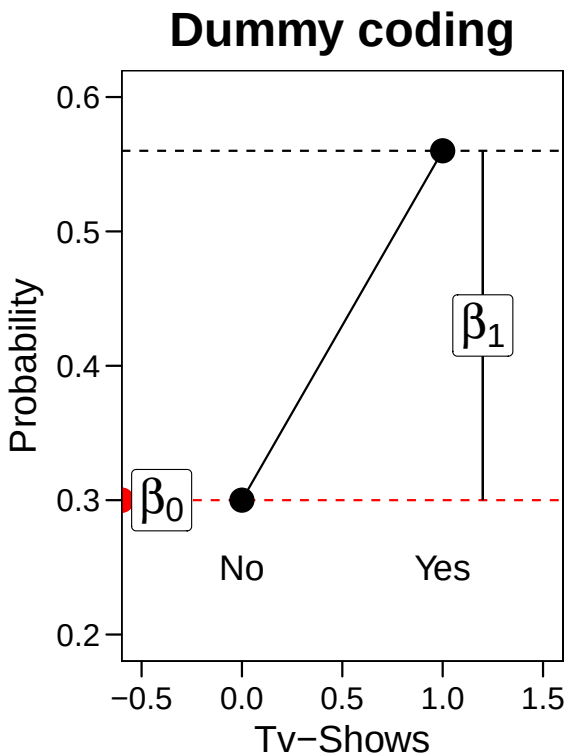
```
dat$tv_shows0 <- ifelse(dat$tv_shows == 0, -0.5, 0.5)
fit <- glm(exam ~ tv_shows0, data = dat, family = binomial(link = "logit"))
# grand mean
mean(c(mean(dat$exam[dat$tv_shows == 1]), mean(dat$exam[dat$tv_shows == 0])))
```

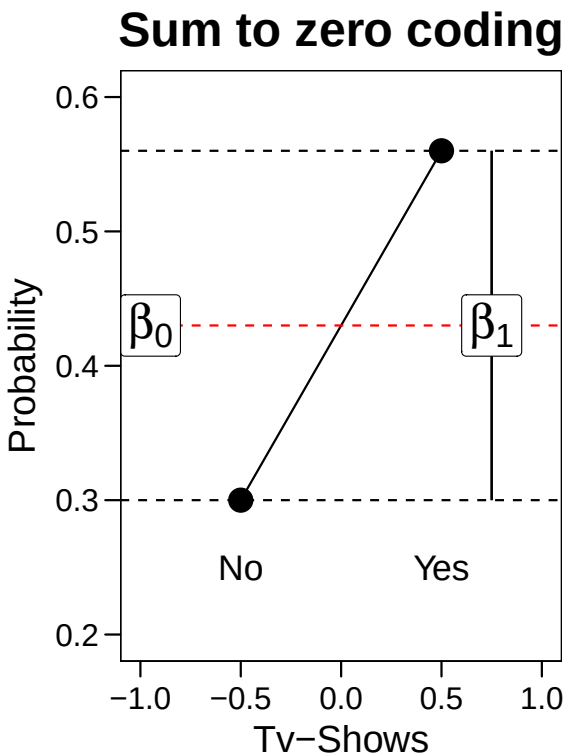
```
## [1] 0.43
```

```
# intercept
plogis(coef(fit)[1])
```

```
## (Intercept)
##   0.4248077
```

Binomial GLM - Categorical predictors

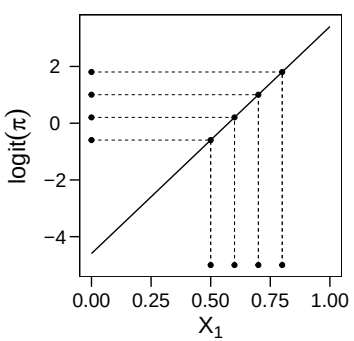


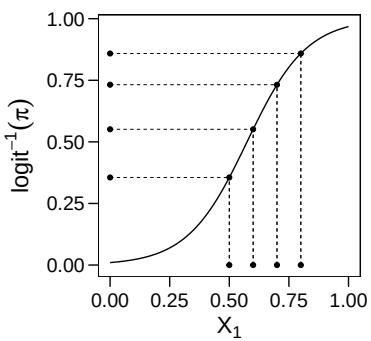


Binomial GLM - Numerical predictors

With numerical predictors the idea is the same as categorical predictors. In fact, categorical predictors are converted into numbers (e.e., 0 and 1 or -0.5 and 0.5). The only caveat is that the effects are linear only the **logit** scale. Thus β_1 is interpreted in the same way as standard linear models only on the link-function scale. For the **binomial**

GLM the β_1 is the increase in the $log(odds(\pi))$ for a unit-increase in the x. In the response (probability) scale, the β_1 is the multiplicative increase in the odds of $y = 1$ for a unit increase in the predictor.





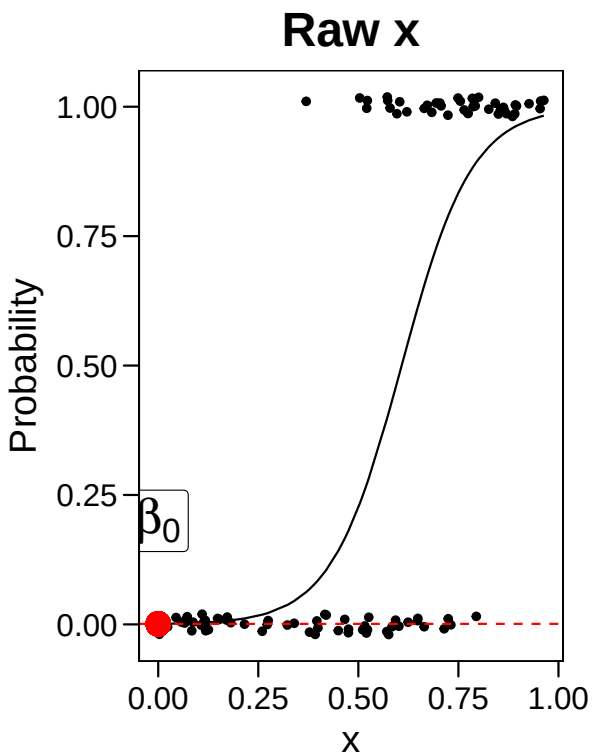
Binomial GLM - Numerical predictors

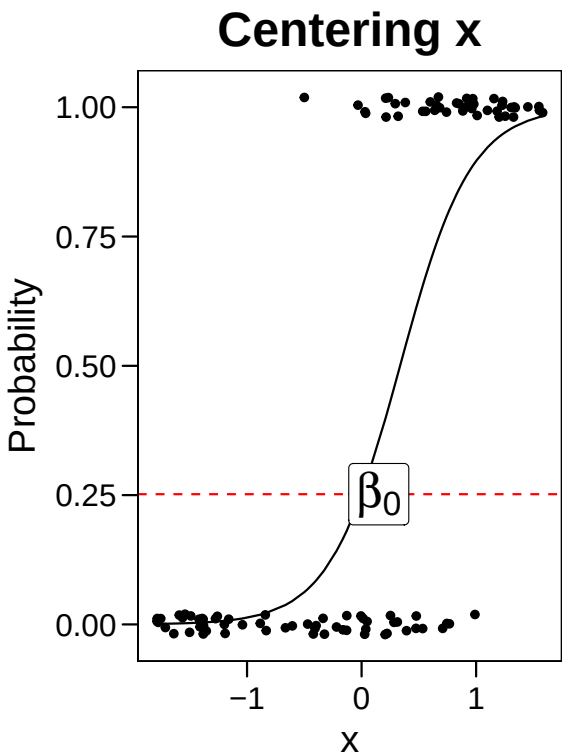
With numerical predictors we could mean-center and or standardize the predictors. With centering (similarly to the categorical example) we change the interpretation of the intercept. Standardizing is helpful to have more meaningful β values. The β_i of a centered predictor is the increase in y for a increase in one standard deviation of x.

$$x_{cen} = x_i - \hat{x}$$

$$x_z = \frac{x_i - \hat{x}}{\sigma_x}$$

Binomial GLM - Numerical predictors





Binomial GLM - Numerical predictors

Let's return to our teddy child example and fitting the proper model:

```
fit_glm <- glm(Depression_pp01 ~ Parental_stress, data = teddy, family = binomial(link = "logit"))
summary(fit_glm)
```

```
##
## Call:
## glm(formula = Depression_pp01 ~ Parental_stress, family = binomial(link = "logit"),
##     data = teddy)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2852  -0.5165  -0.4509  -0.3861   2.3096
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -4.323906   0.690689  -6.260 3.84e-10 ***
## Parental_stress  0.036015   0.009838   3.661 0.000251 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 284.13  on 378  degrees of freedom
## Residual deviance: 271.23  on 377  degrees of freedom
## AIC: 275.23
##
## Number of Fisher Scoring iterations: 5
```

Binomial GLM - Numerical predictors

The `(Intercept)` (β_0) is the probability of having post-partum depression for a mother with parental stress zero (maybe better centering?)

$$p(yes|x = 0) = g^{-1}(\beta_0)$$

```
plogis(coef(fit_glm)["(Intercept)"])
```

```
## (Intercept)
##  0.01307482
```

Binomial GLM - Numerical predictors

The `Parental_stress` (β_1) is the increase in the $log(odds)$ of having the post-partum depression for a unit increase in the parental stress index. If we take the exponential of β_1 we obtain the increase in the $odds$ of having post-partum depression for a unit increase in parental stress index.

```
exp(coef(fit_glm)["Parental_stress"])
```

```
## Parental_stress
##        1.036671
```

Binomial GLM - Numerical predictors

The problem is that, as shown before, the effects are non-linear on the probability scale while are linear on the logit scale. On the logit scale, all differences are constant:

```
pr <- list(c(10, 11), c(50, 51), c(70, 71))

predictions <- lapply(pr, function(x) {
    predict(fit_glm, newdata = data.frame(Parental_stress = x))
})

predictions
```

```
## [[1]]
##         1         2
## -3.963759 -3.927744
##
## [[2]]
##         1         2
## -2.523171 -2.487156
##
## [[3]]
##         1         2
## -1.802877 -1.766862
```

```
# notice that the difference is exactly the Parental_stress parameter
sapply(predictions, diff)
```

```
##         2         2         2
## 0.0360147 0.0360147 0.0360147
```

Binomial GLM - Numerical predictors

While on the probability scale, the differences are not the same. This is problematic when interpreting the results of a Binomial GLM with a numerical predictor.

```
(predictions <- lapply(predictions, plogis))
```

```
## [[1]]
##          1          2
## 0.01863764 0.01930790
##
## [[2]]
##          1          2
## 0.07424969 0.07676350
##
## [[3]]
##         1         2
## 0.1415012 0.1459330
```

```
sapply(predictions, diff)
```

```
##            2            2            2
## 0.0006702661 0.0025138036 0.0044317558
```

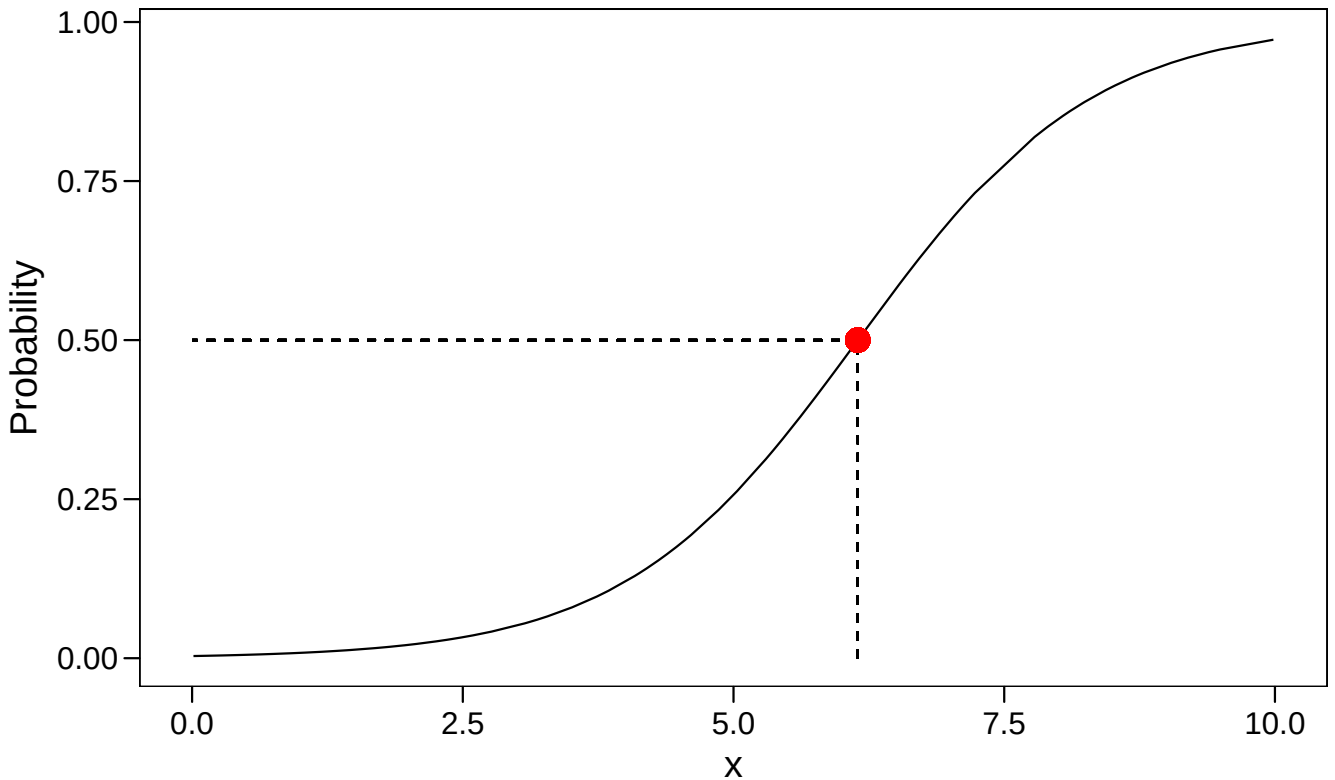
Binomial GLM - Divide by 4 rule

The **divide by 4 rule** is a very easy way to evaluate the effect of a continous predictor on the probability.

Given the non-linearity, the derivative of the logistic function (i.e., the slope) is maximal when predicts probabilities around ~0.5.

In fact, $\beta_i \pi(1-\pi)$ is maximized when $\pi = 0.5$ turning into $\beta_i 0.25$ (i.e., dividing by 4).

Binomial GLM - Divide by 4 rule



Binomial GLM - Predicted probabilities

In a similar way we can use the inverse logit function to find the predicted probability specific values of x. For example, the difference between $p(y = 1|x = 2.5) - p(y = 1|x = 5)$ can be calculated using the model equation:

- $logit^{-1}p(y = 1|x = 2.5) = \frac{e^{\beta_0+\beta_1 2.5}}{1+e^{\beta_0+\beta_1 2.5}}$
- $logit^{-1}p(y = 1|x = 5) = \frac{e^{\beta_0+\beta_1 5}}{1+e^{\beta_0+\beta_1 5}}$
- $logit^{-1}p(y = 1|x = 5) - logit^{-1}p(y = 1|x = 2.5)$

```
coefs <- coef(fit)
plogis(coefs[1] + coefs[2]*5) - plogis(coefs[1] + coefs[2]*2.5)
```

```
## (Intercept)
##   0.2237369
```

Binomial GLM - Predicted probabilities

In R we can use directly the `predict()` function with the argument `type = "response"` to return the predicted probabilities instead of the logits:

```
preds <- predict(fit, newdata = list(x = c(2.5, 5)), type = "response")
preds
```

```
##         1         2
## 0.0329886 0.2567255
```

```
preds[2] - preds[1]
```

```
##         2
## 0.2237369
```

Binomial GLM - Predicted probabilities

I have written the `epredict()` function that extend the `predict()` function giving some useful messages when computing predictions. you can use it with every model and also with multiple predictors.

```
epredict(fit, values = list(x = c(2.5, 5)), type = "response")
```

```
## y ~ -5.693 + 0.926*c(2.5, 5)
```

```
## [1] 0.0329886 0.2567255
```

Binomial GLM - Marginal effects

Marginal effects can be considered very similar to the **divide by 4 rule**. A particularly useful type of marginal effect is the **average marginal effect**. While the **divide by 4** rule estimate the **maximal** difference (in probability scale) according to x, the **average marginal effect** is the average of all slopes (i.e., derivatives) interpreted as the average change in probability scale across all unit increases in x.

```
# calculate the derivative
calc_deriv <- function(b0, b1, x){
    (b1 * exp(b0 + b1 * x)) / (1 + (exp(b0 + b1 * x)))^2
}

coefs <- coef(fit)
dd <- calc_deriv(coefs[1], coefs[2], dat$x)
mean(dd)
```

```
## [1] 0.09338663
```

Binomial GLM - Marginal effects

More efficiently we can do the same using the `margins` package in R:

```
mrg <- margins::margins(fit)
summary(mrg)
```

```
##  factor    AME     SE       z      p  lower  upper
##       x 0.0934 0.0031 29.9339 0.0000 0.0873 0.0995
```

Binomial GLM - Inference

Binomial GLM - Wald tests

The basic approach when doing inference with GLM is intepreting the Wald test of each model coefficients. The Wald test is calculated as follows:

$$z = \frac{\beta_i - \beta_0}{\sigma_{\beta_i}}$$

Calculating the p-value based on a standard normal distribution.

Binomial GLM - Wald-type confidence intervals

Wald-type confidence interval (directly from model summary), where Φ is the cumulative Gaussian function `qnorm()`:

$$95\%CI = \hat{\beta} \pm \Phi(\alpha/2)SE_{\beta}$$

```
(summ <- data.frame(summary(fit)$coefficients))
```

```
##                Estimate Std..Error   z.value     Pr...z..
## (Intercept) -0.6632942  0.2985407 -2.221788 0.0262976067
## tv_shows     1.4170660  0.4254859  3.330465 0.0008670096
```

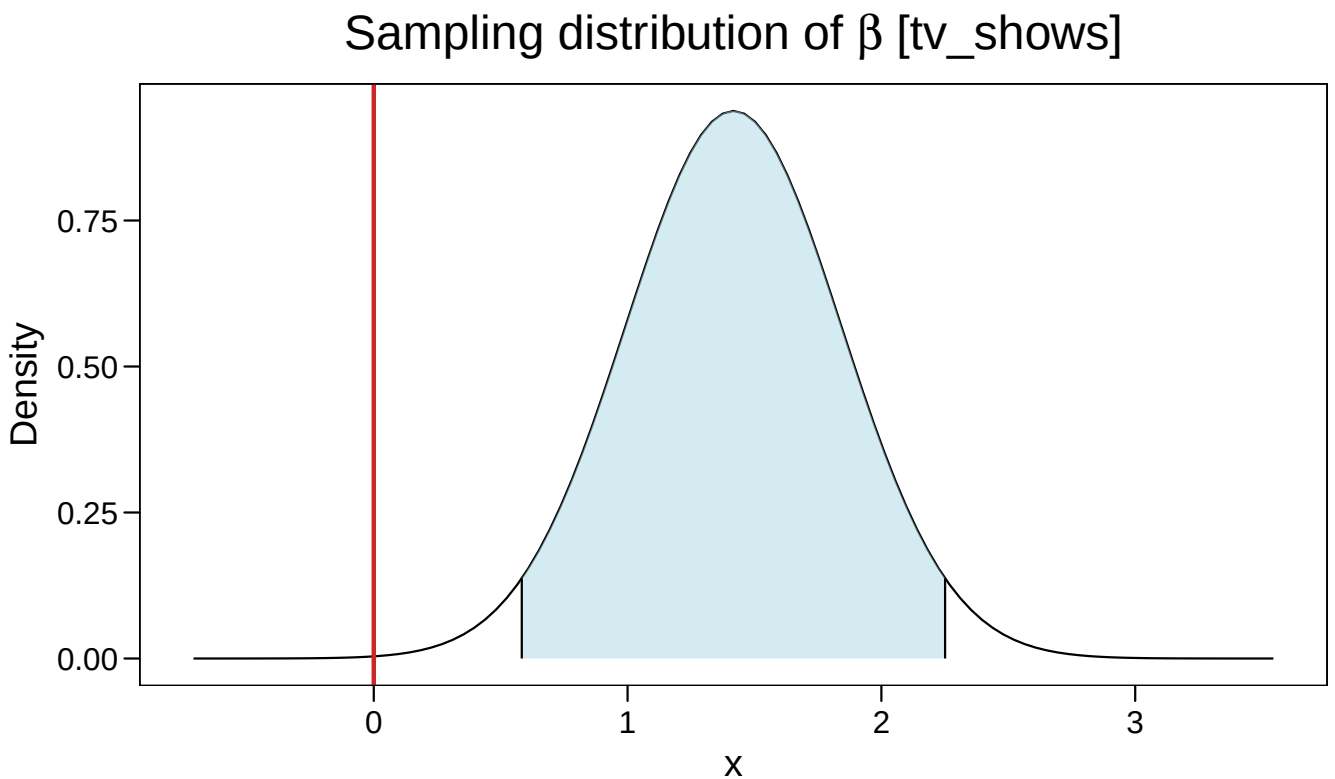
```
# 95% confidence interval
summ$Estimate + qnorm(c(0.025, 0.95))*summ$Std..Error
```

```
## [1] -1.248423  2.116928
```

Binomial GLM - Wald-type confidence intervals

You can also use the `plot_param()` function to represent the sampling distribution and the confidence interval:

```
plot_param(fit, "tv_shows", ci = "z") + mytheme()
```



Binomial GLM - Profile likelihood confidence intervals

The computation is a little bit different and they are not always symmetric:

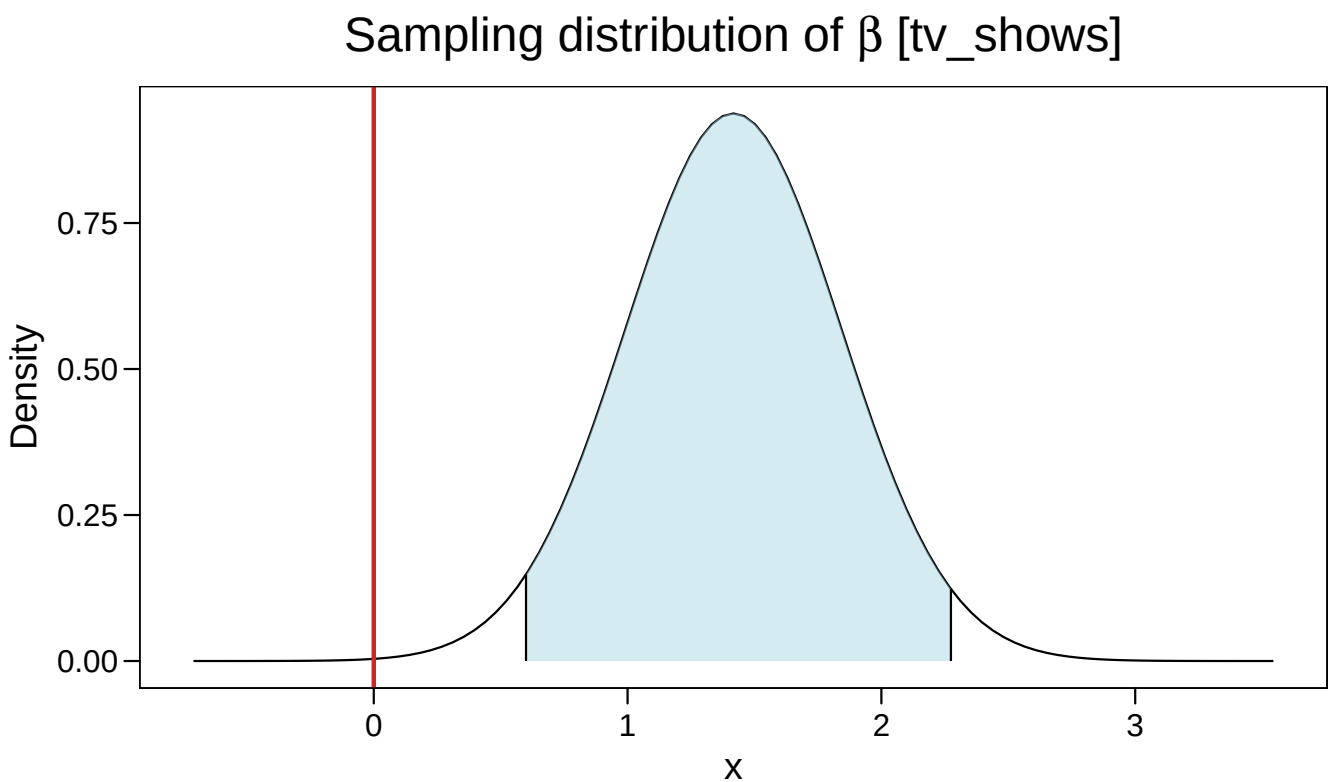
```
# profile likelihood, different from wald type
confint(fit)
```

```
##                  2.5 %      97.5 %
## (Intercept) -1.2711814 -0.09226204
## tv_shows     0.5998267  2.27401606
```

Binomial GLM - Profile likelihood confidence intervals

Again we can use the `plot_param()` function:

```
plot_param(fit, "tv_shows", ci = "profile") + mytheme()
```



Binomial GLM - Confidence intervals

When calculating confidence intervals it is important to consider the link function. In the same way as we compute the inverse logit function on the parameter value, we could revert also the confidence intervals.
IMPORTANT, do not apply the inverse logit on the standard error and then compute the confidence interval.

```
fits <- broom::tidy(fit) # extract parameters as dataframe
fits
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic   p.value
##   <chr>          <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)    -1.52     0.368     -4.12 0.0000380
## 2 tv_shows        2.01     0.469      4.27 0.0000193
```

Binomial GLM - Confidence intervals

```r
b <- fits$estimate[2]
se <- fits$std.error[2]

# correct, wald-type confidence intervals
c(b = exp(b), lower = exp(b - 2*se), upper = exp(b + 2*se))
```

```
##         b     lower     upper
##  7.432749  2.906586 19.007090
```

```r
# correct, likelihood based confidence intervals
exp(confint(fit, "tv_shows"))
```

```
##     2.5 %    97.5 %
##  3.062659 19.530409
```

```r
# wrong wald type
c(b = exp(b), lower = exp(b) - 2*exp(se), upper = exp(b) + 2*exp(se))
```

```
##         b     lower     upper
##  7.432749  4.234493 10.631004
```

Binomial GLM - Confidence intervals

The same principle holds for predicted probabilities. First compute the intervals on the logit scale and then transform-back on the probability scale:

```
fits <- dat |>
    select(tv_shows) |>
    distinct() |>
    add_predict(fit, se.fit = TRUE)

fits$p <- plogis(fits$fit)
fits$lower <- plogis(with(fits, fit - 2*se.fit))
fits$upper <- plogis(with(fits, fit + 2*se.fit))

fits
```

```
## # A tibble: 2 x 7
##   tv_shows    fit se.fit residual.scale     p  lower upper
##      <dbl>  <dbl>  <dbl>          <dbl> <dbl>  <dbl> <dbl>
## 1        1  0.490  0.291              1 0.620 0.477  0.745
## 2        0 -1.52   0.368              1 0.180 0.0951 0.314
```

Binomial GLM - Anova

With multiple predictors, especially with categorical variables with more than 2 levels, we can compute the an anova-like analysis individuating the effect of each predictor. In R we can do this using the `car::Anova()` function. Let's simulate a model with a 2x2 interaction:

```
##   id x1 x2 y
## 1  1  a  c 1
## 2  2  a  d 0
## 3  3  b  c 0
## 4  4  b  d 0
## 5  5  a  c 1
## 6  6  a  d 0
```

We can fit the most complex model containing the two main effects and the interaction[2]:

```
fit_max <- glm(y ~ x1 + x2 + x1:x2, data = dat, family = binomial(link = "logit")) # same as x1 * x2
```

[2] I set the contrasts for the two factors as `contr.sum()/2` that are required for a proper analysis of factorial designs

Binomial GLM - Anova

```
summary(fit_max)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2 + x1:x2, family = binomial(link = "logit"),
##     data = dat)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.066  -1.011  -0.459   1.293   2.146
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.0151     0.2279  -4.454 8.44e-06 ***
## x11           0.5724     0.4559   1.256  0.20924
## x21           1.3565     0.4559   2.976  0.00292 **
## x11:x21      -0.8704     0.9117  -0.955  0.33973
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 144.87  on 119  degrees of freedom
## Residual deviance: 133.54  on 116  degrees of freedom
## AIC: 141.54
##
## Number of Fisher Scoring iterations: 4
```

Binomial GLM - Anova

Then using `car::Anova()`. For each effect we have the χ^2 statistics and the associated p-value. The null hypothesis is that the specific factor did not contribute in reducing the residual deviance.

```
car::Anova(fit_max)
```

```
## Analysis of Deviance Table (Type II tests)
##
## Response: y
##       LR Chisq Df Pr(>Chisq)
## x1      1.0943  1   0.295529
## x2      9.3856  1   0.002187 **
## x1:x2   0.9403  1   0.332195
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Binomial GLM - Model comparison

The table obtained with `car::Anova()` is essentially a model comparison using the Likelihood Ratio test. This can be done using the `anova(...)` function.

$$D = 2(log(\mathcal{L}_{full}) - log(\mathcal{L}_{reduced}))$$
$$D \sim \chi^2_{df_{full} - df_{reduced}}$$

Binomial GLM - Model comparison

To better understanding, the `x2` effect reported in the `car::Anova()` table is a model comparison between a model with `y ~ x1 + x2` and a model without `x2`. The difference between these two model is the unique contribution of `x2` after controlling for `x1`:

```r
fit <- glm(y ~ x1 + x2, data = dat, family = binomial(link = "logit"))
fit0 <- glm(y ~ x1, data = dat, family = binomial(link = "logit"))

anova(fit0, fit, test = "LRT") # ~ same as car::Anova(fit_max)
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ x1
## Model 2: y ~ x1 + x2
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       118     143.86
## 2       117     134.48  1   9.3856 0.002187 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Binomial GLM - Model comparison

The model comparison using `anova()` (i.e., likelihood ratio tests) is limited to nested models thus models that differs only for one term. For example:

```
fit1 <- glm(y ~ x1, data = dat, family = binomial(link = "logit"))
fit2 <- glm(y ~ x2, data = dat, family = binomial(link = "logit"))
fit3 <- glm(y ~ x1 + x2, data = dat, family = binomial(link = "logit"))
```

`fit1` and `fit2` are non-nested because we have the same number of predictors (thus degrees of freedom). `fit3` and `fit1`/`fit2` are nested because `fit3` is more complex and removing one term we can obtain the less complex models.

Binomial GLM - Model comparison

```
anova(fit1, fit2, test = "LRT") # do not works properly
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ x1
## Model 2: y ~ x2
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       118     143.86
## 2       118     135.57  0   8.2914
```

```
anova(fit1, fit3, test = "LRT") # same anova(fit2, fit3)
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ x1
## Model 2: y ~ x1 + x2
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       118     143.86
## 2       117     134.48  1   9.3856 0.002187 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```