Introduction to Generalized Linear Models

Filippo Gambarota

University of Padova

2022/2023

```
devtools::load_all()
library(tidyverse)
library(kableExtra)
library(patchwork)
```

Outline

1. Beyond the Gaussian distribution
2. Generalized Linear Models
3. Relevant distributions
4. Data simulation *[EXTRA]*
5. Binomial GLM
6. Binomial GLM
7. Binomial GLM - Diagnostic

1. Beyond the Gaussian distribution

2. Generalized Linear Models

3. Relevant distributions

4. Data simulation *[EXTRA]*

5. Binomial GLM

6. Binomial GLM

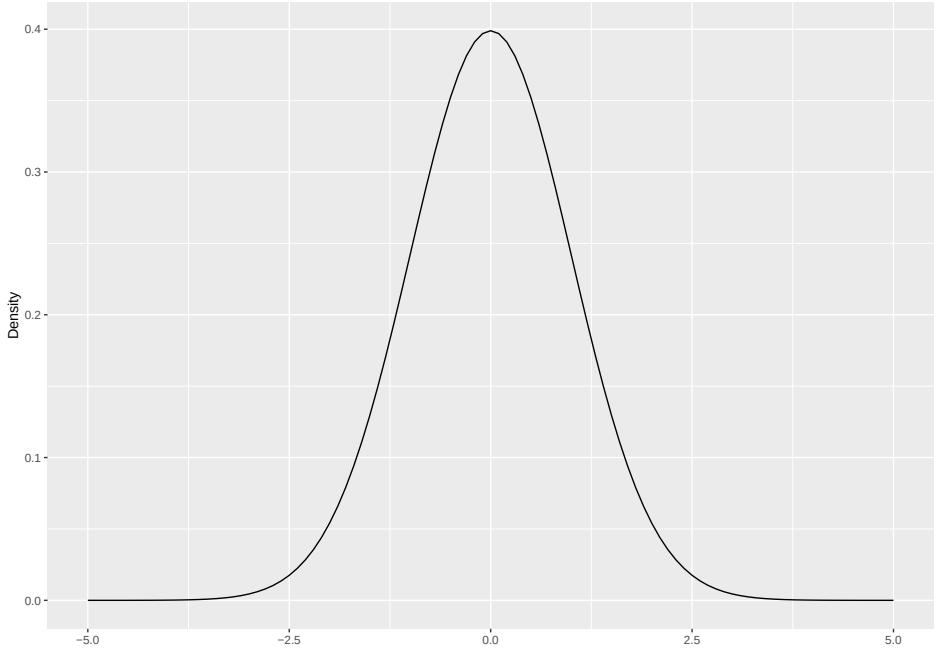
7. Binomial GLM - Diagnostic

8. Binomial GLM - Parameter Intepretation

Quick recap about Gaussian distribution

- function
- parameters
- support

“‘r ggnorm(0, 1) “‘

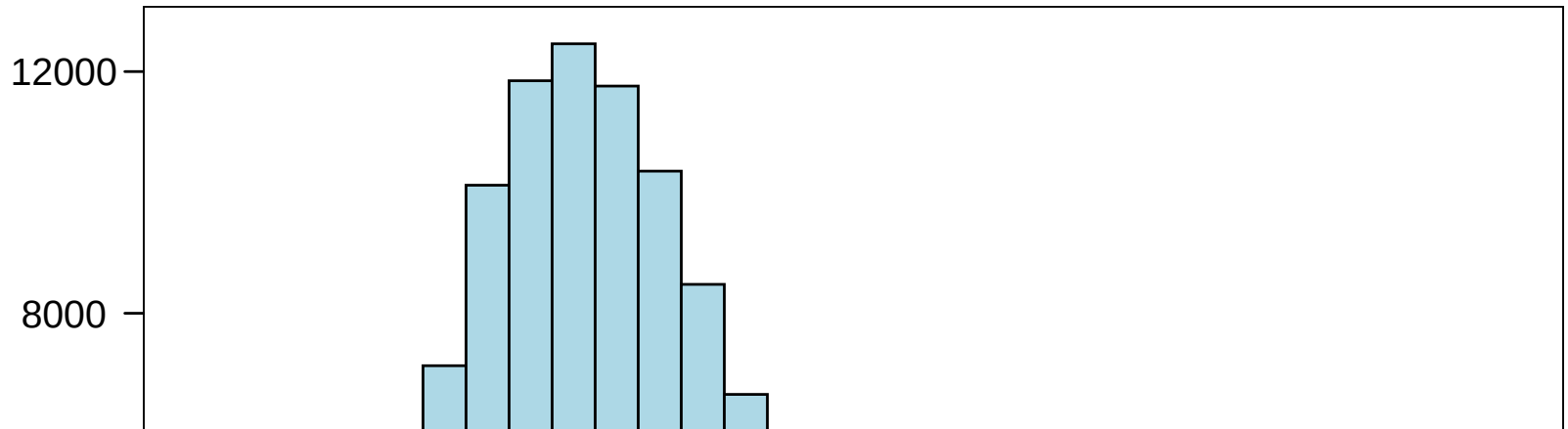


But not always gaussian-like variables!

Reaction times

Measuring reaction times during a cognitive task. Non-negative and proably skewed data.

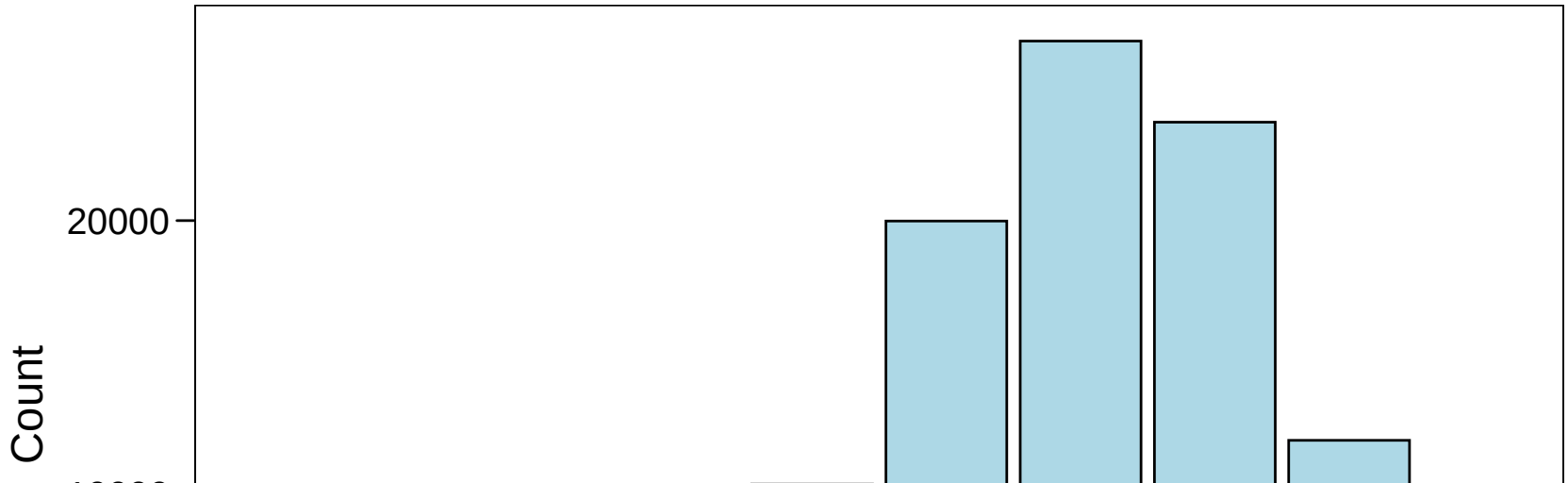
```
dat <- data.frame(
    x = rgamma(1e5, 9, scale = 0.5)*100
)

dat |>
    ggplot(aes(x = x)) +
    geom_histogram(fill = "lightblue",
                   color = "black") +
    xlab("Reaction Times (ms)") +
    ylab("Count") +
    mytheme()
```



Binary outcomes

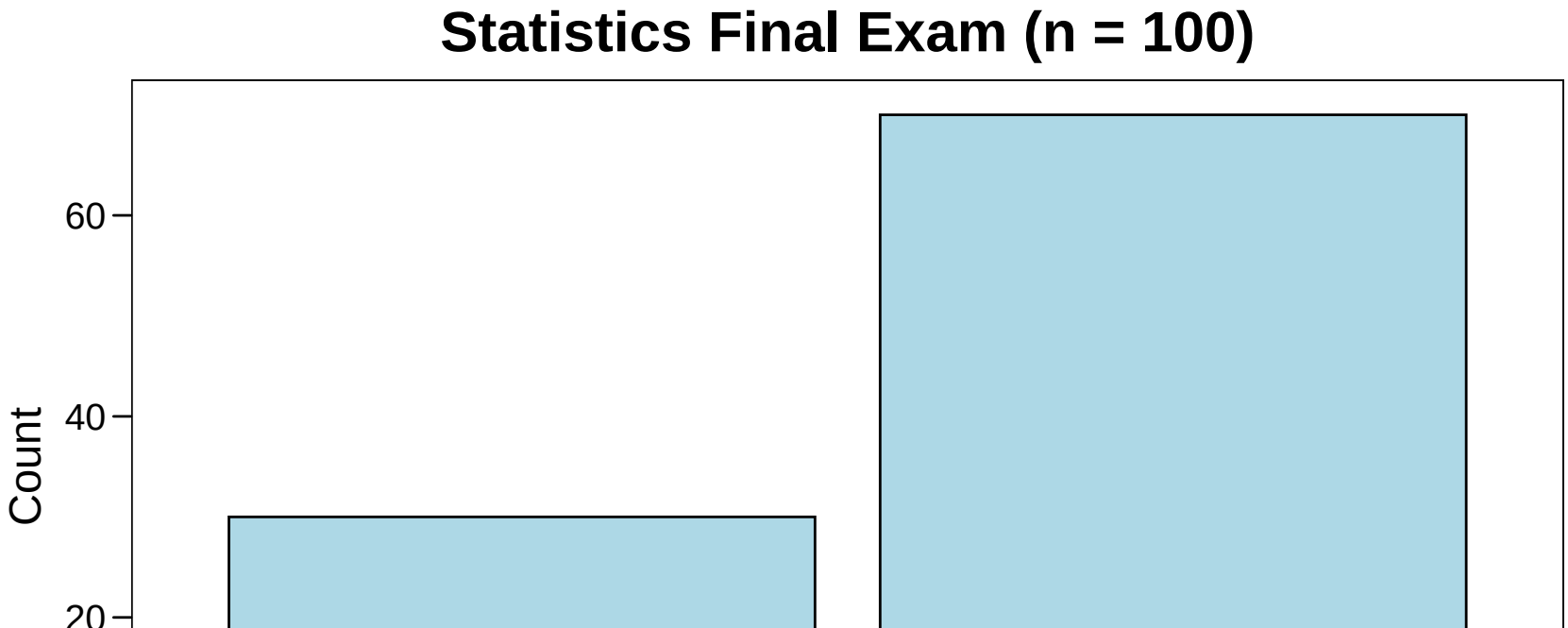
Counting the number of people passing the exam out of the total.
Discrete and non-negative. A series of binary (i.e., *bernoulli*) experiments.

```
dat <- data.frame(x = rbinom(1e5, 10, 0.7))

dat |>
    ggplot(aes(x = factor(x))) +
    geom_bar(fill = "lightblue",
             color = "black") +
    xlab("Number of success out of 10 trials") +
    ylab("Count") +
    mytheme()
```



Binary outcomes

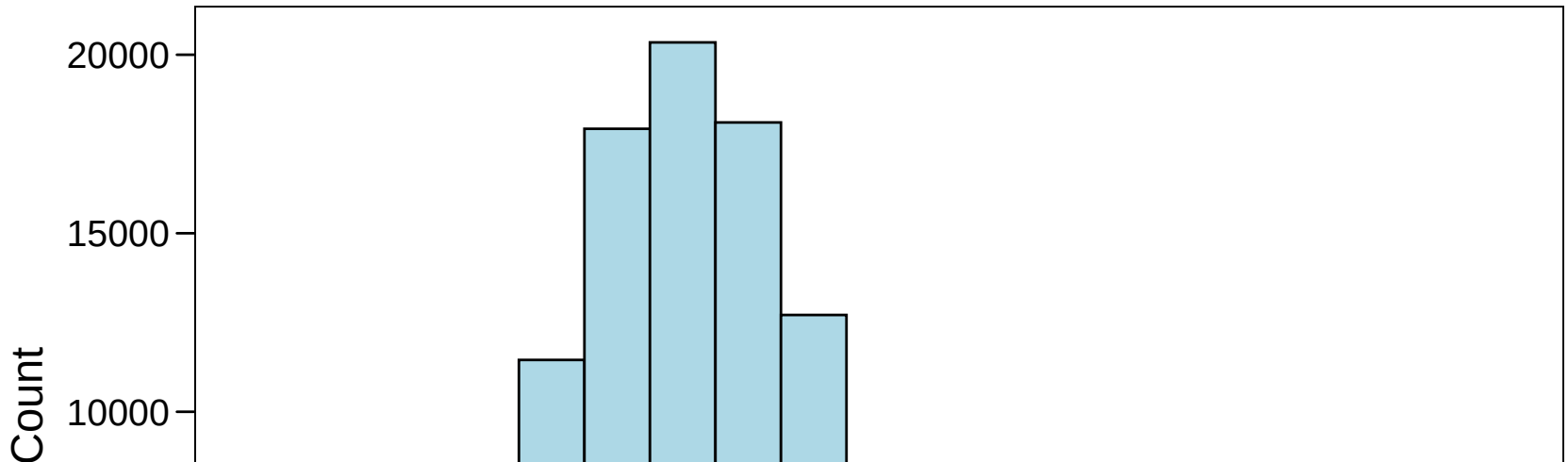
```
dat <- data.frame(y = c(70, 30), x = c("Passed", "Failed"))

dat |>
    ggplot(aes(x = x, y = y)) +
    geom_col(color = "black",
             fill = "lightblue") +
    ylab("Count") +
    mytheme() +
    theme(axis.title.x = element_blank()) +
    ggtitle("Statistics Final Exam (n = 100)")
```



Counts

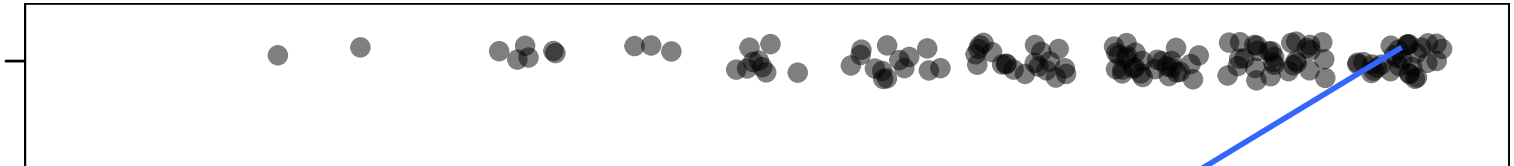
Counting the number of new patients per week. Discrete and non-negative values.

```
dat <- data.frame(x = rpois(1e5, 15))

dat |>
    ggplot(aes(x = x)) +
    geom_histogram(binwidth = 2,
                   color = "black",
                   fill = "lightblue") +
    xlab("Number of new patients per week") +
    ylab("Count") +
    mytheme()
```



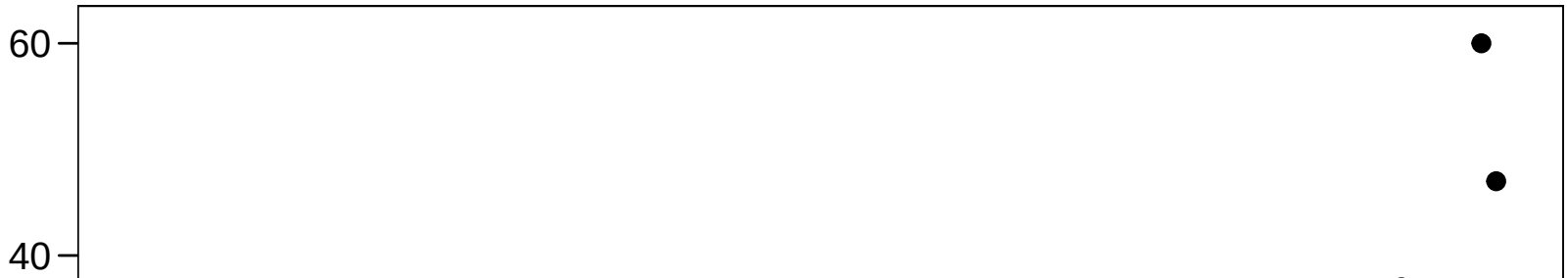
Should we use a linear model for these variables?

```r
# y = number of exercises solved in 1 semester
# x = percentage of attended lectures

n <- 30
x <- rep(0:10, each = n)
b0 <- 0.01
b1 <- 0.8
y <- rbinom(length(x), 1, plogis(qlogis(b0) + b1*x))

dat <- data.frame(x, y)

dat |>
    ggplot(aes(x = x, y = y)) +
    geom_hline(yintercept = 0, linetype = "dashed", col = "red") +
    geom_point(size = 3,
               alpha = 0.5,
               position = position_jitter(height = 0.03)) +
    geom_smooth(method = "lm",
                se = F) +
    mytheme() +
    scale_x_continuous(breaks = 0:10) +
    xlab("Questions difficulty") +
    ylab("Probability of passing the exam")
```



Should we use a linear model for these variables?

```r
# y = number of exercises solved in 1 semester
# x = percentage of attended lectures

n <- 30
x <- runif(n, 0, 1)
b0 <- 0
b1 <- 4
y <- rpois(n, exp(b0 + b1*x))

dat <- data.frame(x, y)

dat |>
    ggplot(aes(x = x*100, y = y)) +
    geom_hline(yintercept = 0, linetype = "dashed", col = "red") +
    geom_point(size = 3) +
    geom_smooth(method = "lm",
                se = F) +
    mytheme() +
    xlab("Percetage of attended lectures") +
    ylab("Solved exercises")
```



Should we use a linear model for these variables?

```
fit <- lm(y ~ x, data = dat)

dfit <- data.frame(
    fitted = fitted(fit),
    residuals = residuals(fit)
)

qqn <- dfit |>
    ggplot(aes(sample = residuals)) +
    stat_qq() +
    stat_qq_line() +
    xlab("Theoretical Quantiles") +
    ylab("Residuals") +
    mytheme()

res_fit <- dfit |>
    ggplot(aes(x = fitted, y = residuals)) +
    geom_point() +
    mytheme() +
    ylab("Residuals") +
    xlab("Fitted")
qqn + res_fit
```

A new class of models

- We need that our model take into account the **features of our response variable**
- We need a model that, **with appropriate transformation**, keep **properties of standard linear models**
- We need a model that is **closer to the true data generation process**

Let's switch to Generalized Linear Models!

1. Beyond the Gaussian distribution
2. Generalized Linear Models
3. Relevant distributions
4. Data simulation *[EXTRA]*
5. Binomial GLM
6. Binomial GLM
7. Binomial GLM - Diagnostic
8. Binomial GLM - Parameter Intepretation

Main references

add books here

General idea

- models that assume distributions other than the normal distributions
- models that considers non-linear relationships

Recipe for a GLM

- **Random Component**
- **Systematic Component**
- **Link Function**

Random Component

Systematic Component

Link Function

1. Beyond the Gaussian distribution
2. Generalized Linear Models
3. Relevant distributions
4. Data simulation *[EXTRA]*
5. Binomial GLM
6. Binomial GLM
7. Binomial GLM - Diagnostic
8. Binomial GLM - Parameter Intepretation

Binomial distribution

Poisson distribution

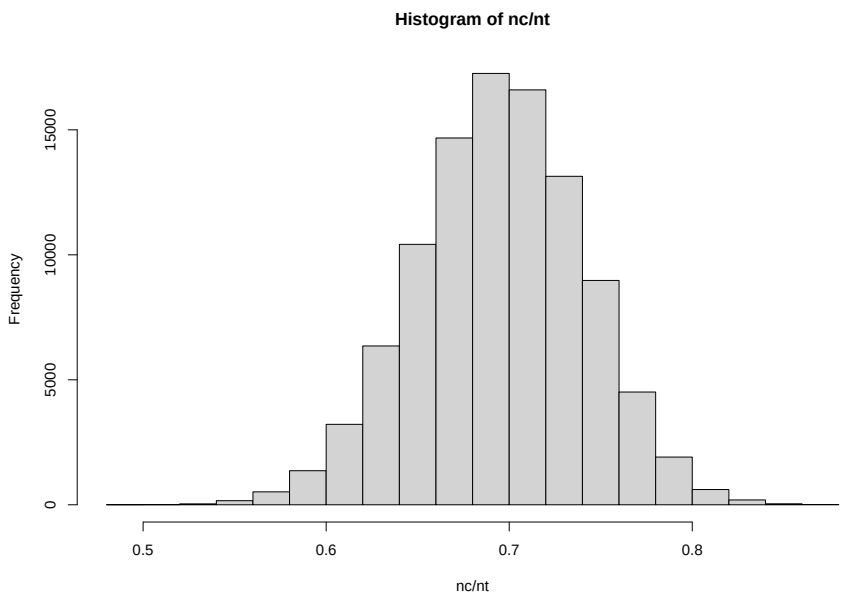
1. Beyond the Gaussian distribution
2. Generalized Linear Models
3. Relevant distributions
4. Data simulation *[EXTRA]*
5. Binomial GLM
6. Binomial GLM
7. Binomial GLM - Diagnostic
8. Binomial GLM - Parameter Intepretation

Data simulation *[EXTRA]*

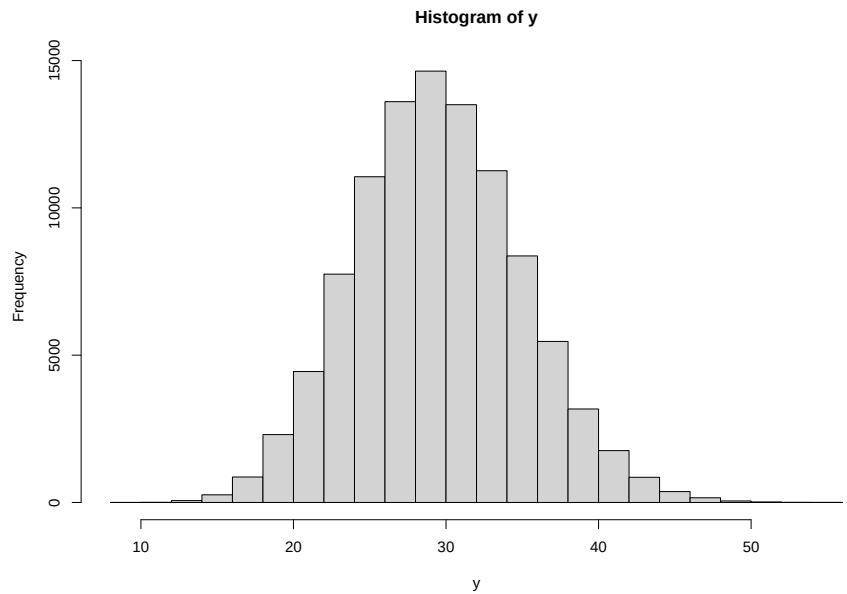
- During the course we will try to simulate some data. Simulating data is an amazing education tool to understand a statistical model.
- By simulating from a **generative model** we are doing **Monte Carlo Simulations** [1]

Data simulation *[EXTRA]*

```r
n <- 1e5 # number of experiments
nt <- 100 # number of subjects
p <- 0.7 # probability of success
nc <- rbinom(n, nt, p)
```




```r
n <- 1e5 # number of subjects
lambda <- 30 # mean/variance
y <- rpois(n, lambda)
```



1. Beyond the Gaussian distribution
2. Generalized Linear Models
3. Relevant distributions
4. Data simulation *[EXTRA]*
5. Binomial GLM
6. Binomial GLM
7. Binomial GLM - Diagnostic
8. Binomial GLM - Parameter Intepretation

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

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  19  50
##   1    19  31  50
##   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 0.19 0.50
##   1   0.19 0.31 0.50
##   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:

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

- 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] 1.631579
```

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

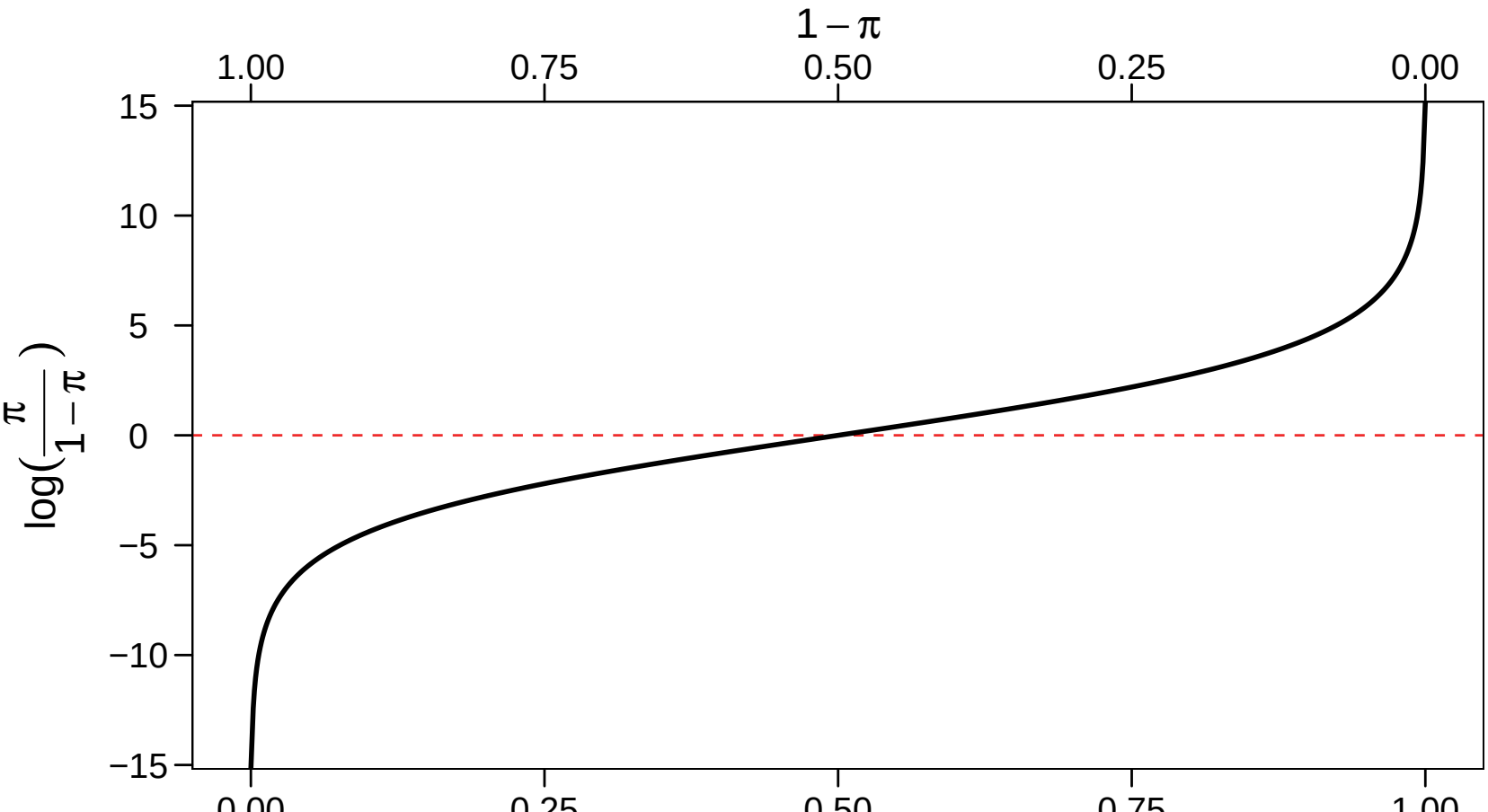
Example: Passing the exam - Odds Ratio

```
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] 2.66205
```

Why using these measure?

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



1. Beyond the Gaussian distribution
2. Generalized Linear Models
3. Relevant distributions
4. Data simulation *[EXTRA]*
5. Binomial GLM
6. Binomial GLM
7. Binomial GLM - Diagnostic
8. Binomial GLM - Parameter Intepretation

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:

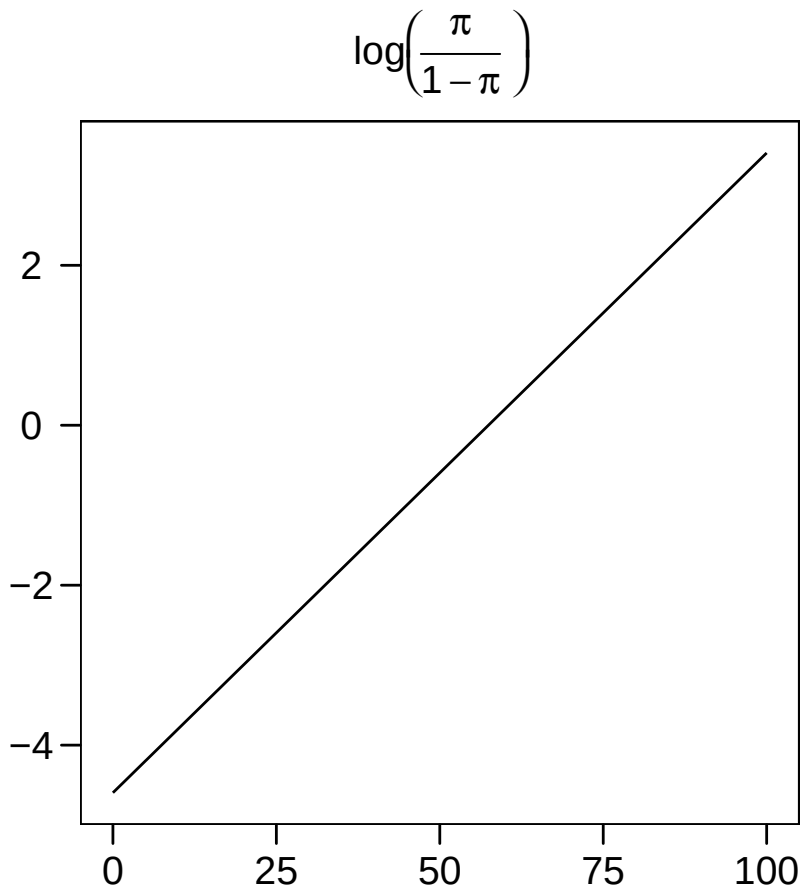
$$log\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 X_1 + ...\beta_p X_p$$

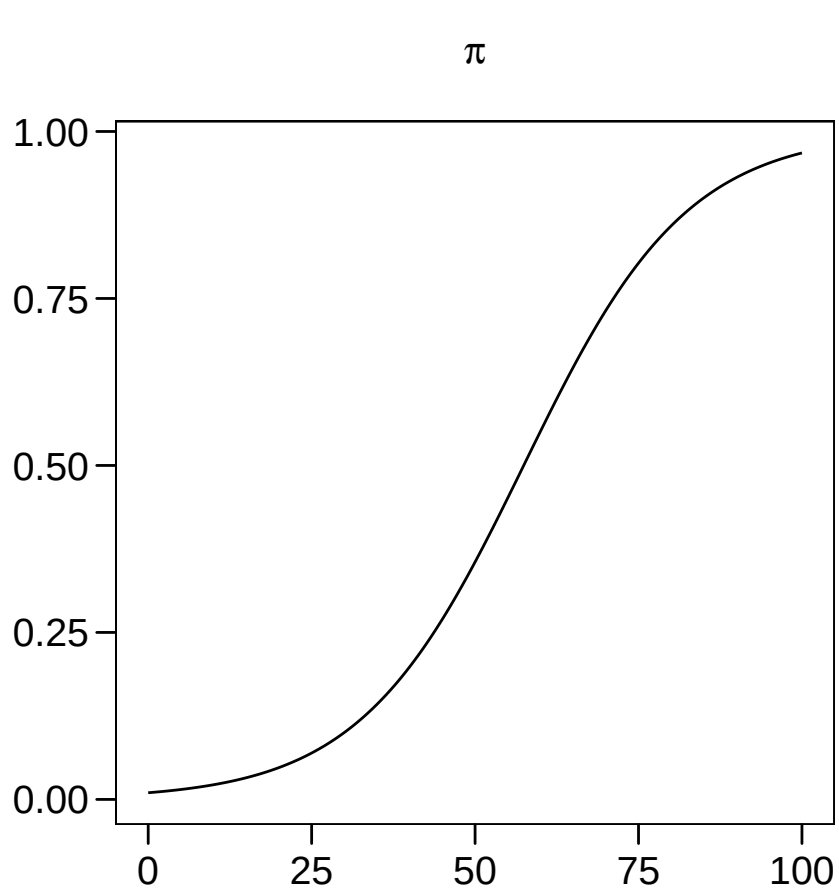
The inverse of the **logit** maps again the probability into the $[0, 1]$ range:

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

Binomial GLM - Logit Link

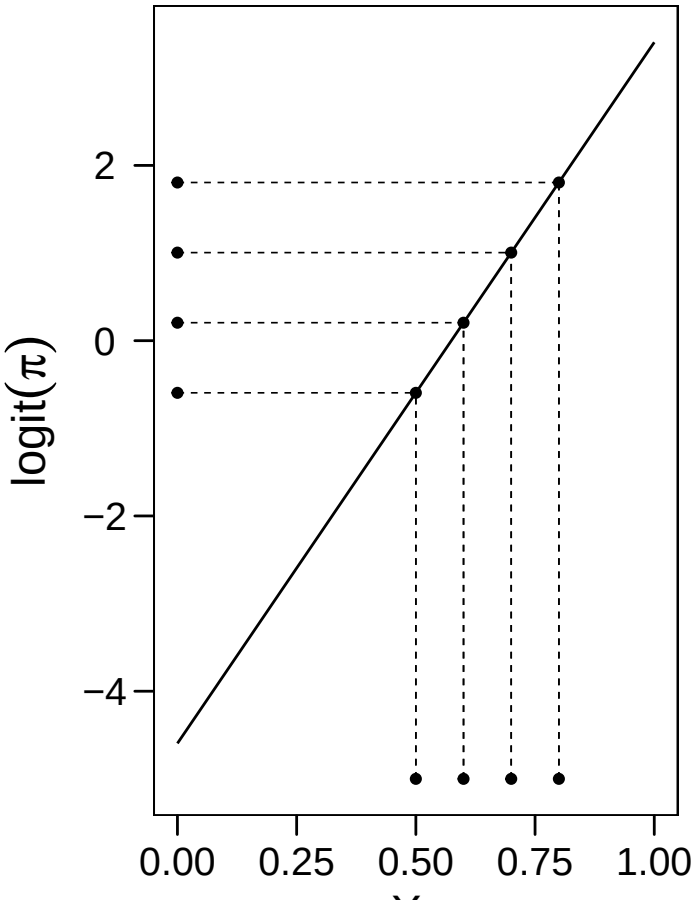
Thus with a single numerical predictor x the relationship between x and π in non-linear on the probability scale but linear on the logit scale.

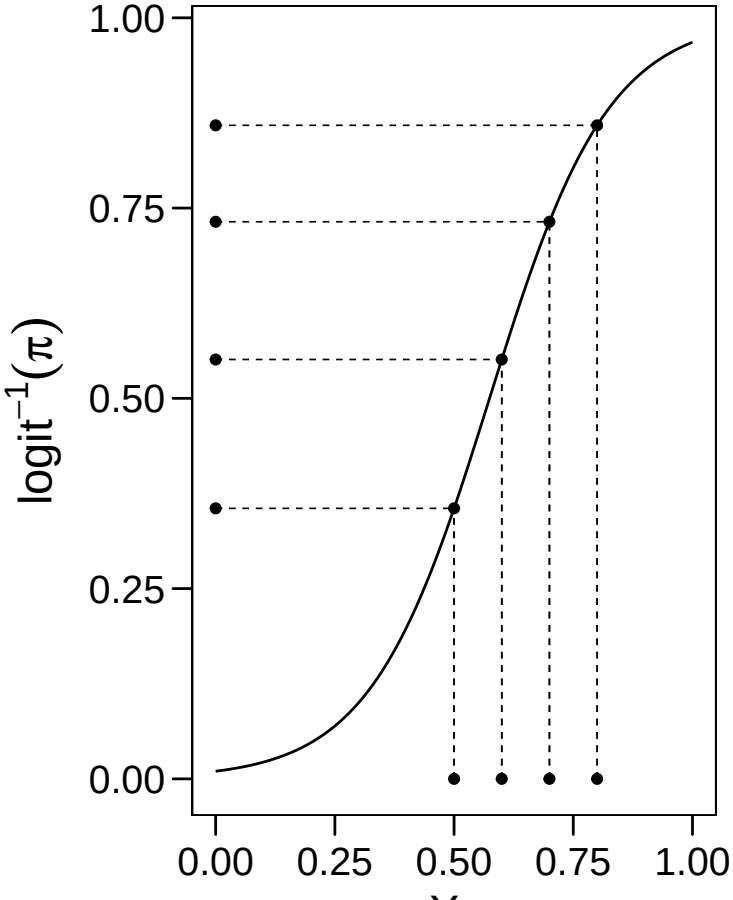




Binomial GLM - Logit Link

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



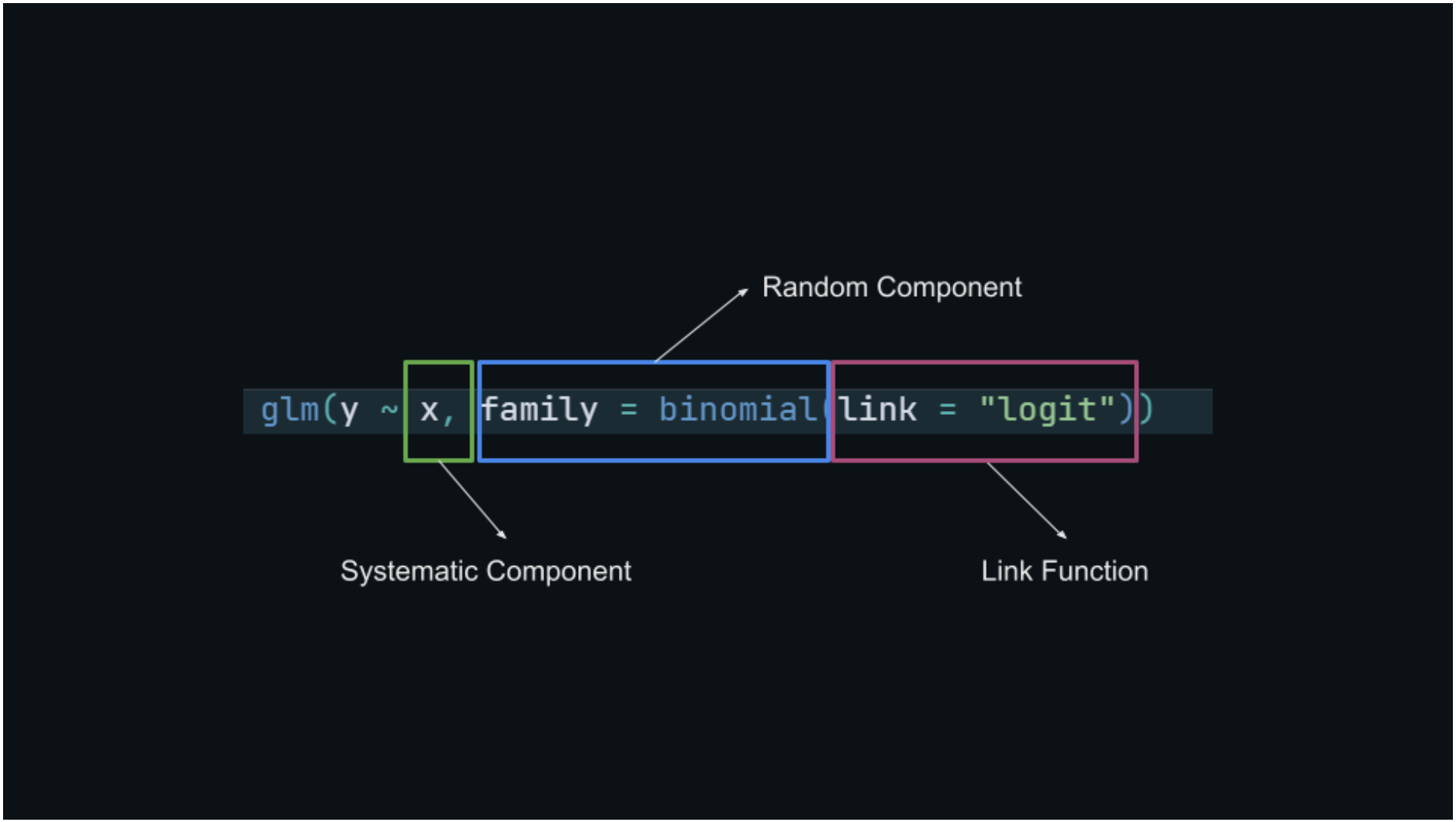


Binomial GLM - Model fitting

ciao

Binomial GLM - Model fitting in R

```
knitr::include_graphics("img/glm-fitting-R.png")
```



Binomial GLM - Model fitting in R

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

```
dat |>
    trim_df()
```

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

```
# TODO add table here
```

Binomial GLM - Intercept only model

Let's start from the simplest model where the systematic part is only the intercept:

```
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
## -1.093  -1.093  -1.093   1.264   1.264
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.2007     0.2010  -0.998    0.318
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 137.63  on 99  degrees of freedom
## Residual deviance: 137.63  on 99  degrees of freedom
## AIC: 139.63
##
## Number of Fisher Scoring iterations: 3
```

Binomial GLM - Intercept only model

When fitting an intercept-only model, the parameter is the average value of the y variable. In our case, we are fitting a model $logis(\pi) = \beta_0$. Thus using the inverse of the link function we obtain the average y on the response scale:

In R the `logis` is the function to work with the logit transformation. `plogis()` is $logit^{-1}(\pi)$ and `qlogis` is $logit(\pi)$

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

```
## [1] 0.45
```

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

```
##     logit.(Intercept) inv-logit.(Intercept)
##             -0.2006707             0.4500000
```

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
## -1.4294  -0.7760  -0.7760   0.9448   1.6414
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.0460     0.3224  -3.244 0.001178 **
## tv_shows      1.6213     0.4368   3.712 0.000205 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 137.63  on 99  degrees of freedom
## Residual deviance: 122.65  on 98  degrees of freedom
## AIC: 126.65
```

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
## 5.059829
```

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

```
## [1] 5.059829
```

1. Beyond the Gaussian distribution
2. Generalized Linear Models
3. Relevant distributions
4. Data simulation *[EXTRA]*
5. Binomial GLM
6. Binomial GLM
7. Binomial GLM - Diagnostic
8. Binomial GLM - Parameter Intepretation

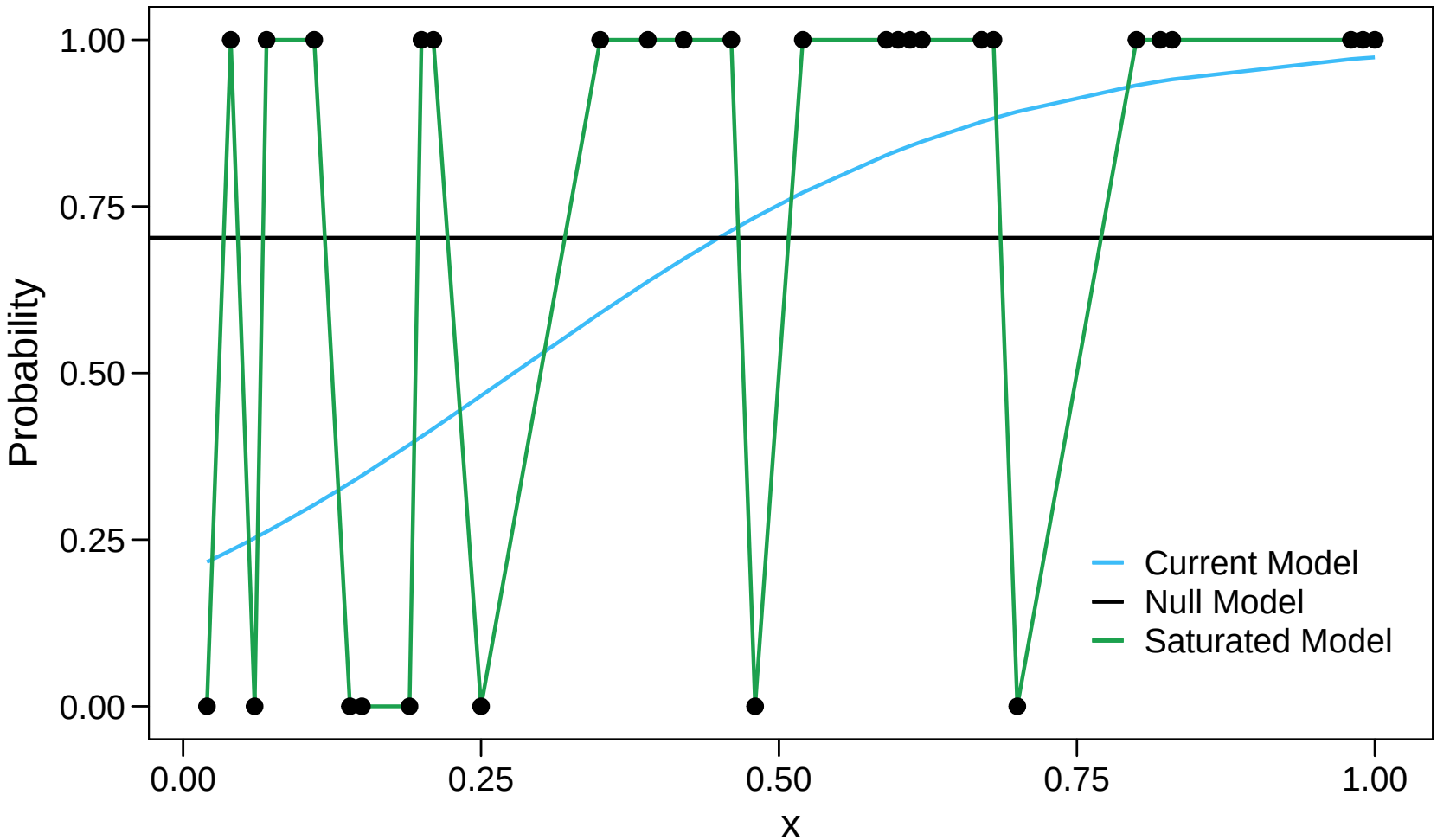
Binomial GLM - Residual Deviance

The residual deviance of a GLM is conceptually similar to the residual standard deviation of a standard linear model.

$$D = -2loglik(\beta)$$

The distance between a perfect model (i.e., saturated) and the current model can be considered as a goodness of fit measure. SPOILER the deviance is also useful to calculate a R^2 like measure comparing the null model with the actual model.

Binomial GLM - Residual Deviance[1]



1. Beyond the Gaussian distribution
2. Generalized Linear Models
3. Relevant distributions
4. Data simulation *[EXTRA]*
5. Binomial GLM
6. Binomial GLM
7. Binomial GLM - Diagnostic
8. 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 [2], [3]
- 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:

```
model.matrix(fit) |>
    as.data.frame() |>
    trim_df()
```

```
##   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:

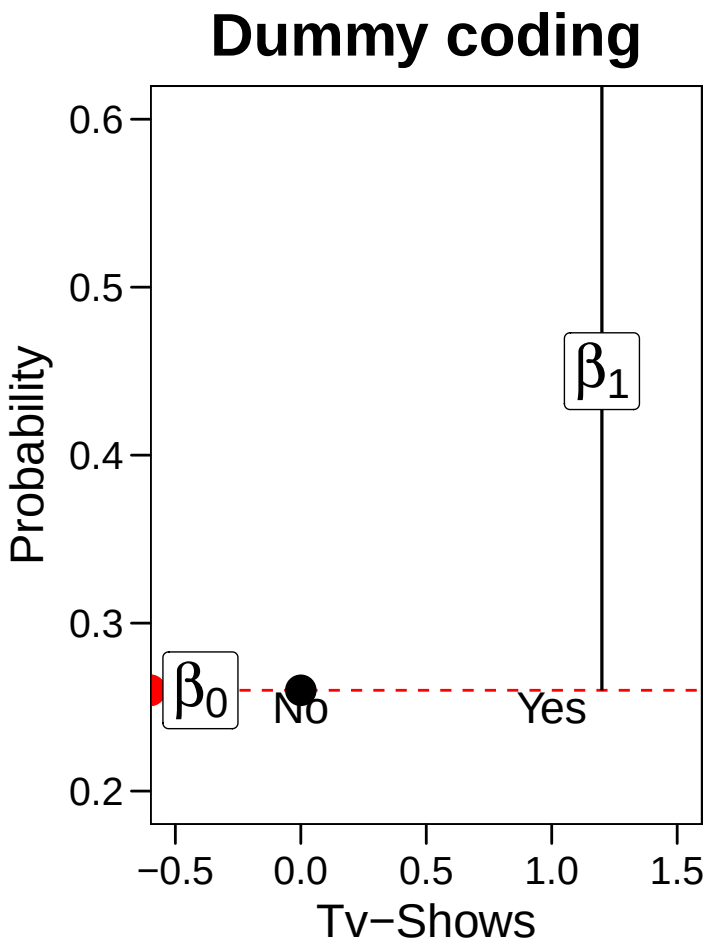
```
dat$tv_shows0 <- ifelse(dat$tv_shows == 0, -0.5, 0.5)
fit <- glm(exam ~ tv_shows0, data = dat, family = binomial(link = "logit"))
# grand mean
mean(c(mean(dat$exam[dat$tv_shows == 1]), mean(dat$exam[dat$tv_shows == 0])))
```

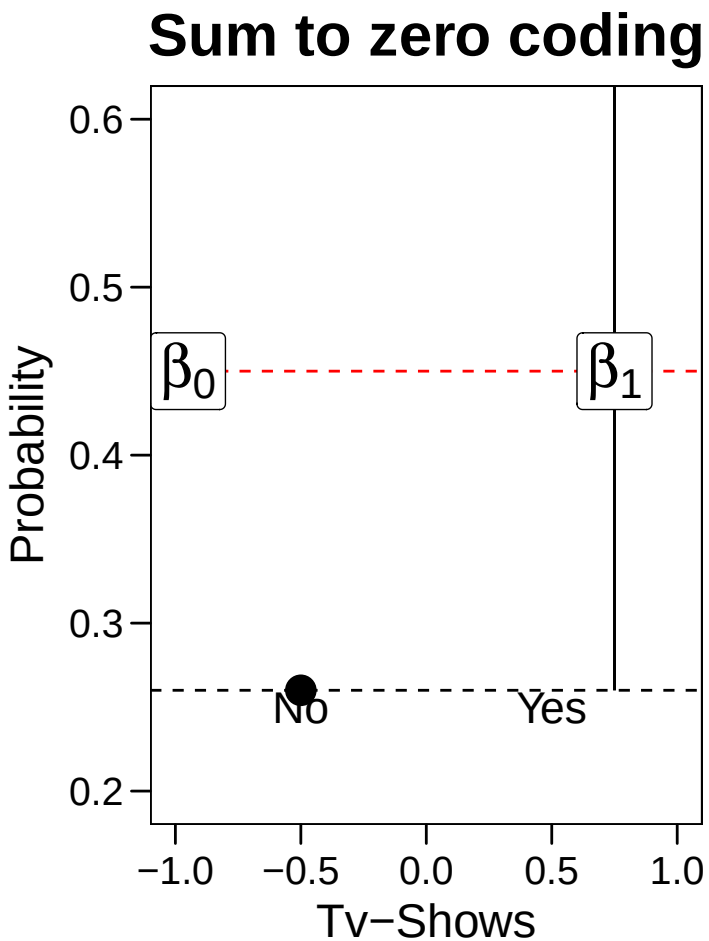
```
## [1] 0.45
```

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

```
## (Intercept)
##   0.4414444
```

Binomial GLM - Categorical predictors

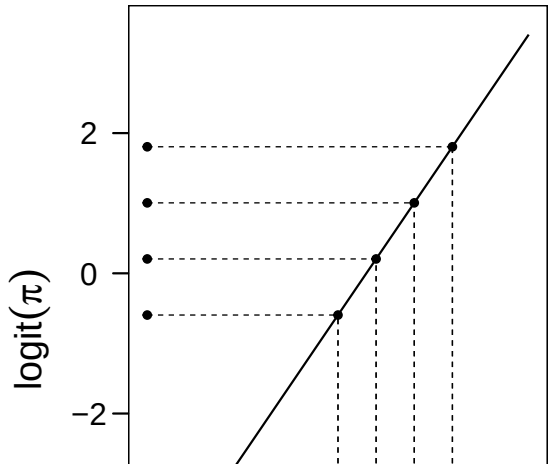


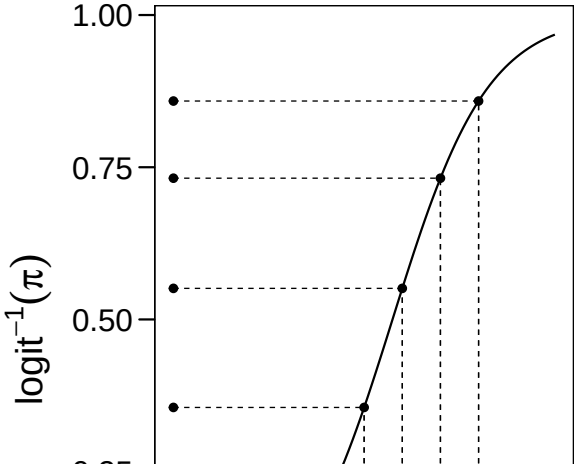


Binomial GLM - Numerical predictors

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.

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

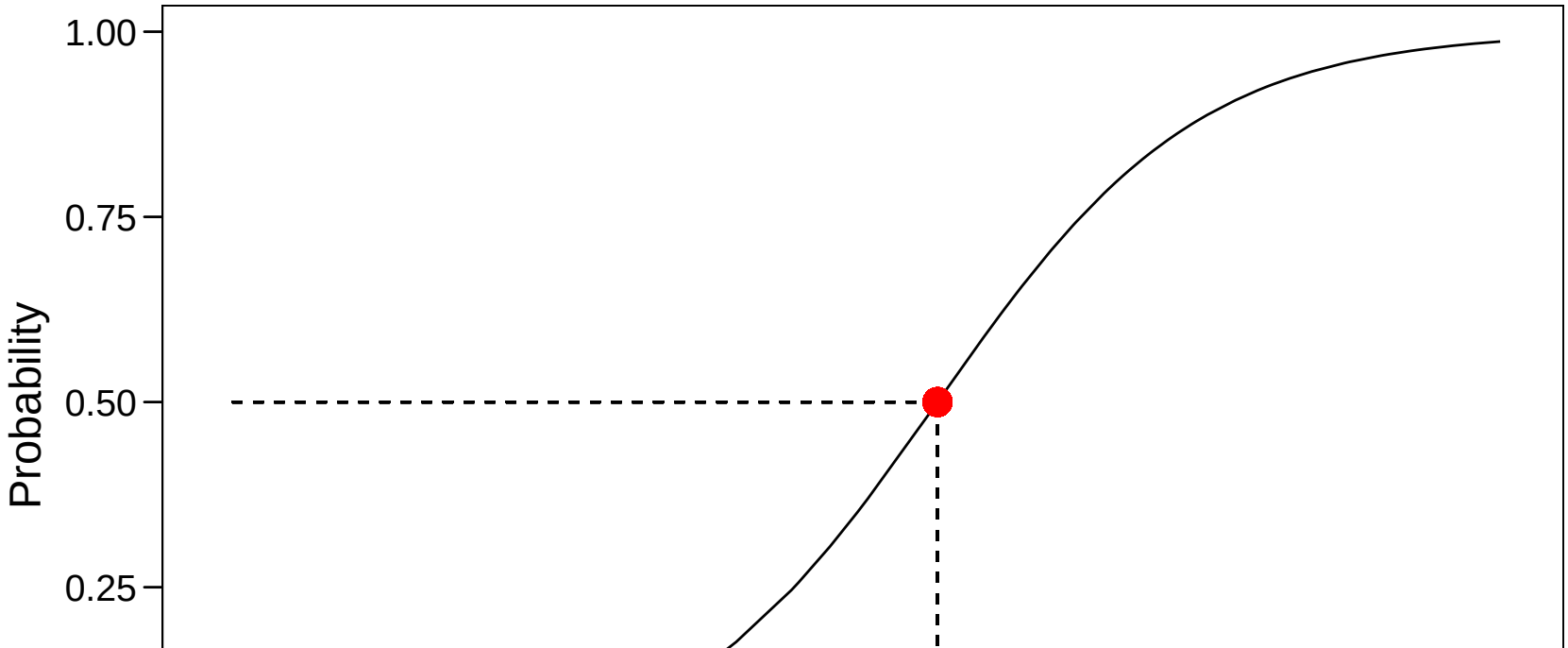
$$x_z = \frac{x_i - \hat{x}}{sigma_x}$$

Binomial GLM - Numerical predictors

```
x <- runif(100, 0, 1)
y <- rbinom(length(x), 1, plogis(qlogis(0.01) + x * 8))
dat <- data.frame(y, x)
dat$x0 <- scale(dat$x)
fit <- glm(y ~ x, data = dat, family = binomial(link = "logit"))
fit0 <- glm(y ~ x0, data = dat, family = binomial(link = "logit"))

raw <- dat |>
    add_predict(fit, type = "response") |>
    mutate(prl = qlogis(pr)) |>
    ggplot(aes(x = x, y = pr)) +
    geom_point(aes(x = x, y = y),
               position = position_jitter(height = 0.02)) +
    geom_line() +
    geom_point(x = 0, y = plogis(coef(fit)[1]),
               size = 5, col = "red") +
    geom_hline(yintercept = plogis(coef(fit)[1]),
               linetype = "dashed",
               col = "red") +
    mytheme() +
    xlab("x") +
    ylab("Probability") +
    annotate("label", x = 0, y = 0.20,
             label = latex2exp::TeX("$\\beta_0$"),
             size = 8) +
    ggtitle("Raw x")

centered <- dat |>
    add_predict(fit, type = "response") |>
    mutate(prl = qlogis(pr)) |>
    ggplot(aes(x = x0, y = pr)) +
    geom_point(aes(x = x0, y = y),
               position = position_jitter(height = 0.02)) +
    geom_line() +
```

Binomial GLM - Divide by 4 rule

The **divide by 4 rule** is a very easy but powerful way to rapidly 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 maximixed when $\pi = 0.5$ turning into $\beta_i 0.25$ (i.e., dividing by 4):



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)
##   0.3154952
```

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
## 0.04700512 0.36250032
```

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

```
##         2
## 0.3154952
```

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 ~ -5.454 + 0.978*c(2.5, 5)
```

```
## [1] 0.04700512 0.36250032
```

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
# TODO add references
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] 0.08286471
```

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 0.0829 0.0029 28.8360 0.0000 0.0772 0.0885
```

1. Beyond the Gaussian distribution
2. Generalized Linear Models
3. Relevant distributions
4. Data simulation *[EXTRA]*
5. Binomial GLM
6. Binomial GLM
7. Binomial GLM - Diagnostic
8. Binomial GLM - Parameter Intepretation

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. We can calculate also the confidence interval for the model coefficients:

$$95\%CI = \beta_i \pm \Phi(\alpha/2)\sigma_{\beta_i}$$

Binomial GLM - Confidence intervals

Different types of confidence intervals

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

```
##                  2.5 %    97.5 %
## (Intercept) -7.9954441 -3.629176
## x            0.6707661  1.395292
```

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:

```
n <- 100
n_watching <- n/2 # just for simplicity
p_pass_yes <- 0.7
p_pass_no <- 0.3

dat <- data.frame(
    tv_shows = rep(c(1, 0), each = n_watching)
)

b0 <- 0.3 # P(passing|not watching)
b1 <- 3.5 # odds ratio between watching and not watching

# pn_from_or(0.3, 3.5) # P(passing|watching)

dat$exam <- rbinom(n, 1, plogis(qlogis(b0) + log(b1)*dat$tv_shows))
fit <- glm(exam ~ tv_shows, data = dat, family = binomial(link = "logit"))
```

```
fits <- broom::tidy(fit)

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

# correct, wald-type confidence intervals
```

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 -0.0800  0.283              1 0.480 0.344 0.619
## 2        0 -1.27    0.341              1 0.220 0.125 0.358
```

Binomial GLM - Model comparison

Binomial GLM - Anova

1. Beyond the Gaussian distribution
2. Generalized Linear Models
3. Relevant distributions
4. Data simulation *[EXTRA]*
5. Binomial GLM
6. Binomial GLM
7. Binomial GLM - Diagnostic
8. Binomial GLM - Parameter Intepretation

Binomial GLM - Marginal effects

When plotting a binomial GLM the most useful way is plotting the marginal probabilities and standard errors/confidence intervals for a given combination of predictors. Let's make an example for:

- simple GLM with 1 categorical/numerical predictor
- GLM with 2 numerical/categorical predictors
- GLM with interaction between numerical and categorical predictors

Binomial GLM - Marginal effects

A general workflow could be:

- fit the model
- use the `predict()` function giving the grid of values on which computing predictions
- calculating the confidence intervals
- plotting the results

Everything can be simplified using some packages to perform each step and returning a plot:

- `allEffects()` from the `effects()` package (return a base R plot)
- `ggeffect()` from the `ggeffect()` package (return a `ggplot2` object)
- `plot_model` from the `sjPlot` package (similar to `ggeffect()`)

Binomial GLM - 1 categorical predictor

In this situation we can just plot the marginal probabilities for each level of the categorical predictor. Plotting our exam dataset:

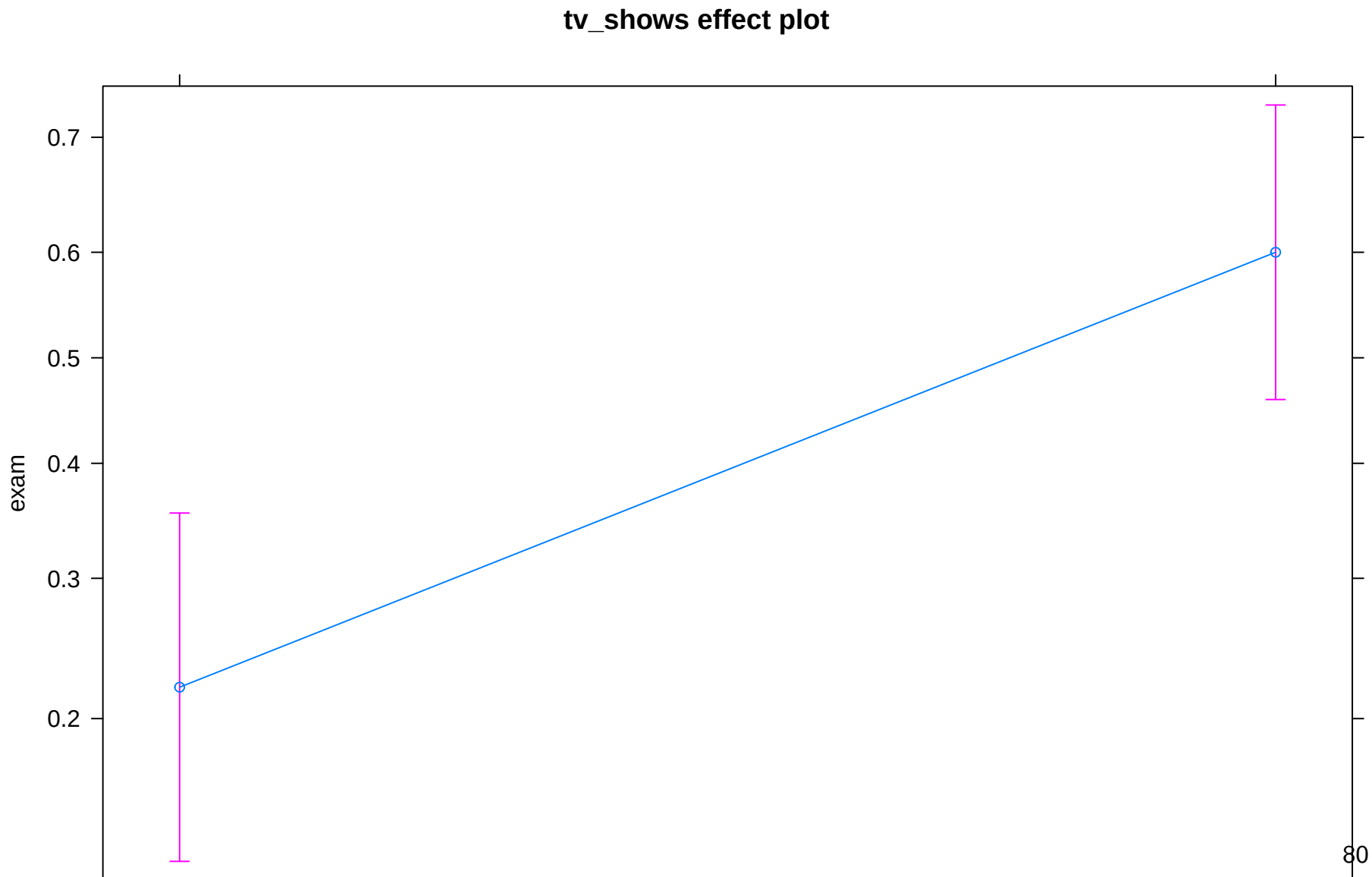
```
n <- 100
n_watching <- n/2 # just for simplicity
p_pass_yes <- 0.7
p_pass_no <- 0.3

dat <- data.frame(
    tv_shows = rep(c(1, 0), each = n_watching)
)

b0 <- 0.3 # P(passing|not watching)
b1 <- 3.5 # odds ratio between watching and not watching

# pn_from_or(0.3, 3.5) # P(passing|watching)

dat$exam <- rbinom(n, 1, plogis(qlogis(b0) + log(b1)*dat$tv_shows))
dat$tv_shows <- factor(dat$tv_shows)

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

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

fits |>
    mutate(tv_shows = ifelse(tv_shows == 0, "No", "Yes"),
           lower = plogis(fit - se.fit * 2),
           upper = plogis(fit + se.fit * 2)) |>
    ggplot(aes(x = tv_shows, y = plogis(fit), ymin = lower, ymax = upper)) +
```

Binomial GLM - 1 numerical predictor

```
x <- runif(100, 0, 1)
y <- rbinom(length(x), 1, plogis(qlogis(0.01) + x * 8))
dat2 <- data.frame(y, x)

fit2 <- glm(y ~ x, data = dat, family = binomial(link = "logit"))

dat2 |>
    add_predict(fit2, se.fit = TRUE) |>
    mutate(p = plogis(fit),
           lower = plogis(fit - 2 * se.fit),
           upper = plogis(fit + 2 * se.fit)) |>
    ggplot(aes(x = x, y = p, ymin = lower, ymax = upper)) +
    geom_line() +
    geom_ribbon(alpha = 0.3) +
    mytheme() +
    ylab("Probability")
```

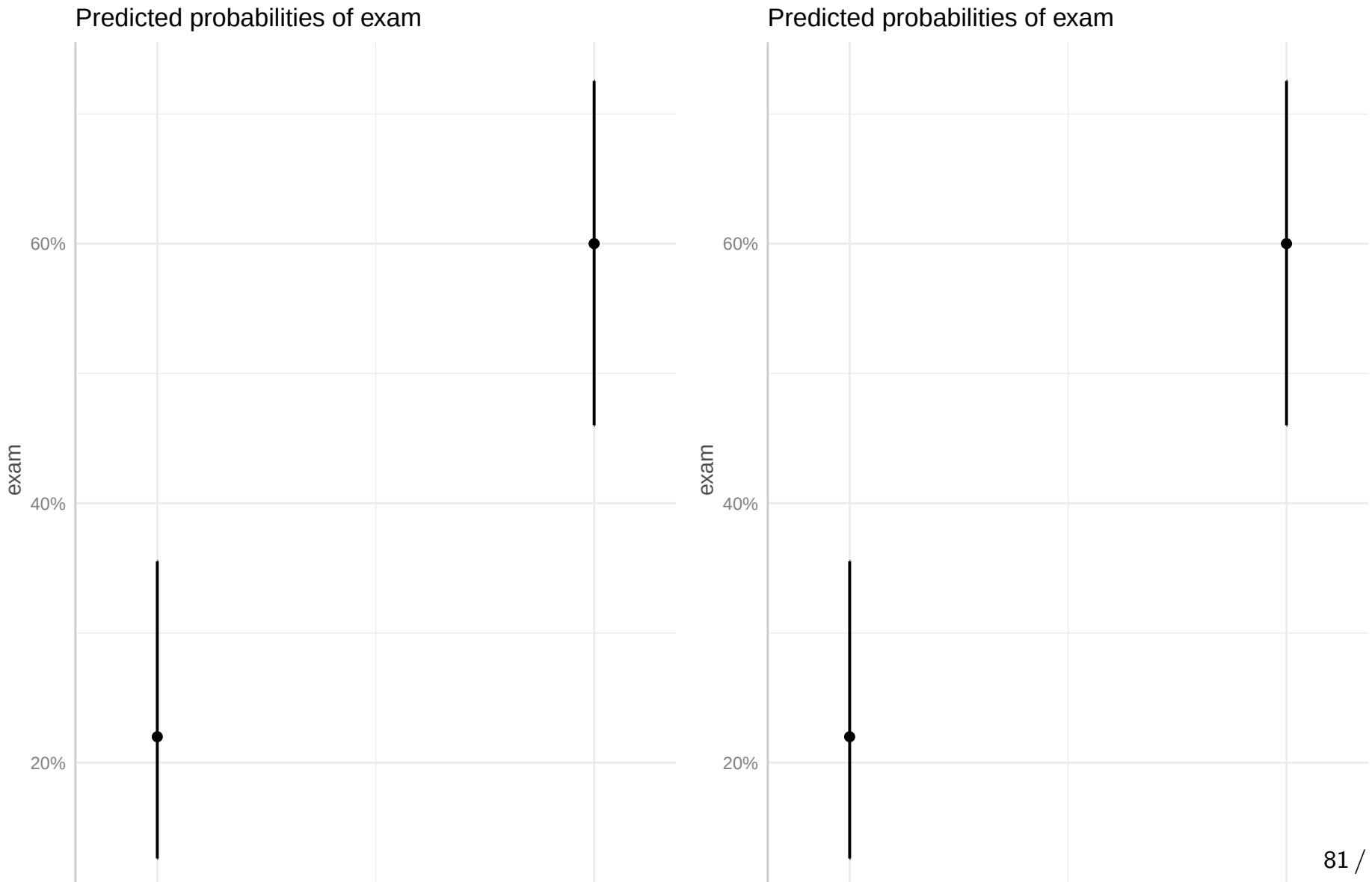
Binomial GLM - `allEffects()`

```
par(mfrow = c(1,2))
plot(effects::allEffects(fit))
```



Binomial GLM - ggeffect()/plot_model()

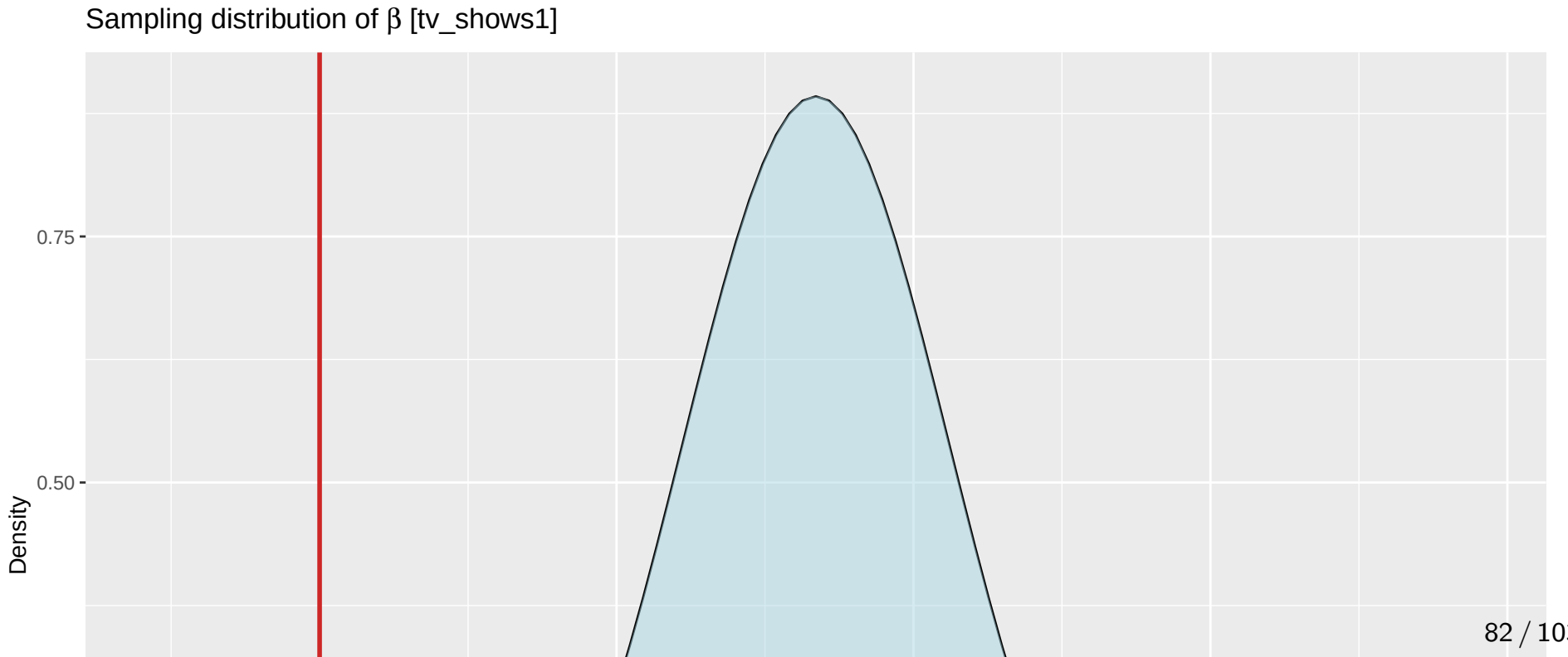
```
plot(ggeffects::ggeffect(fit)[[1]]) + plot(ggeffects::ggeffect(fit)[[1]])
```



Binomial GLM - Plotting coefficients

Sometimes could be useful to plot the estimated sampling distribution of a coefficient. For example, we can plot the `tv_shows` effect on our example. I've written the `plot_param()` function that directly create a basic-plot given the model and the coefficient name. The plot highlight the null value and the 95% Wald confidence interval:

```
plot_param(fit, "tv_shows1")
```



1. Beyond the Gaussian distribution
2. Generalized Linear Models
3. Relevant distributions
4. Data simulation *[EXTRA]*
5. Binomial GLM
6. Binomial GLM
7. Binomial GLM - Diagnostic
8. Binomial GLM - Parameter Intepretation

Binomial GLM - Diagnostic

The diagnostic for GLM is similar to standard linear models. Some areas are more complicated for example residual analysis and goodness of fit. We will see:

- Residuals
  - Types of residuals
  - Residual deviance
- Classification accuracy
- R^2

Binomial GLM - Residuals

As for standard linear models there are different types of residuals:

- raw residuals
- pearson residuals
- standardized pearson residuals
- studentized residuals
- deviance residuals

Binomial GLM - Raw Residuals

Raw residuals are the simplest (and mostly unuseful) type of residuals. They are calculated as in standard regression as:

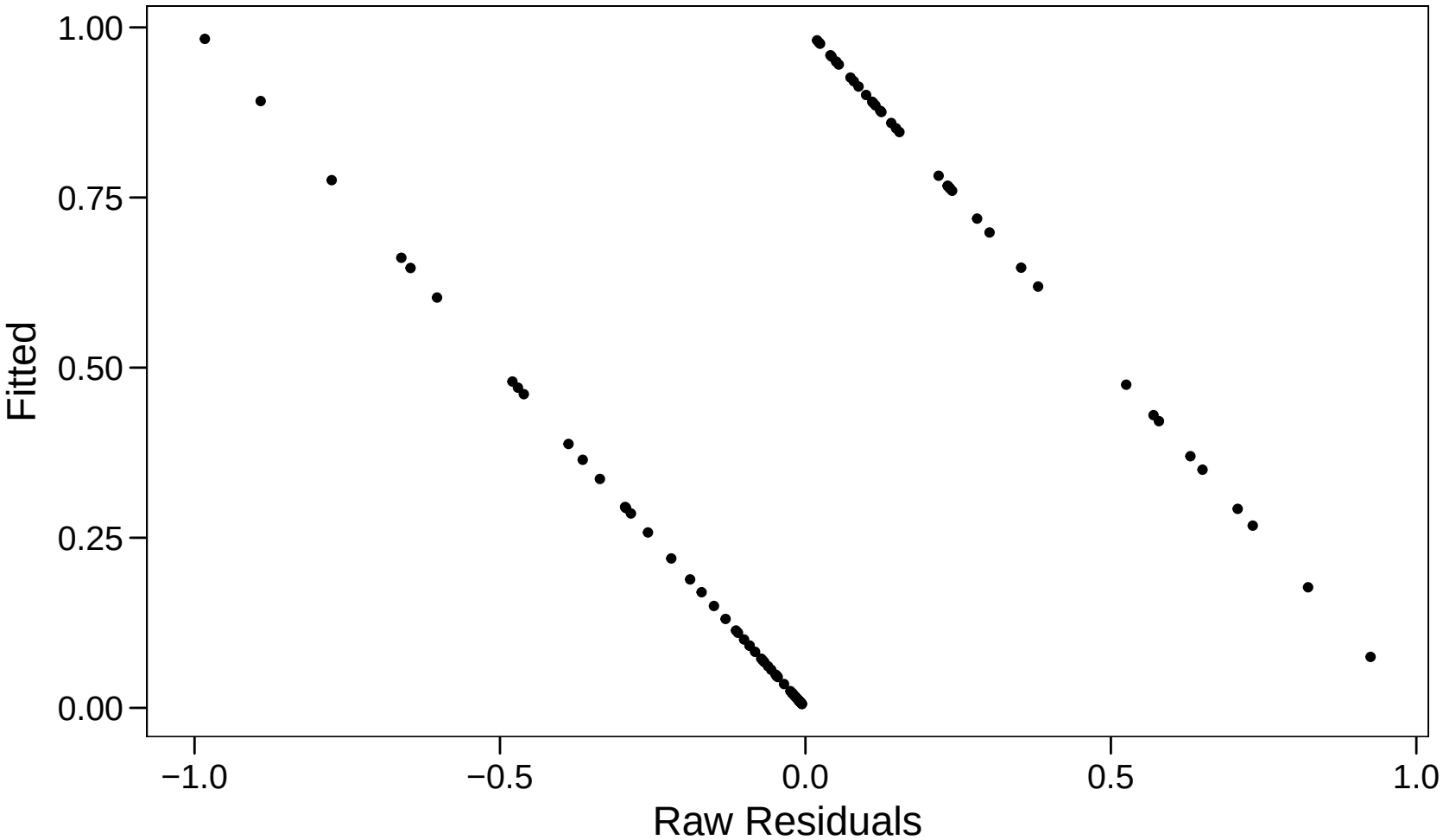
$$r_i = y_i - \hat{y}_i$$

```
# equivalent to residuals(fit, type = "response")
ri <- dat$y - fitted(fit)
ri[1:5]
```

```
## [1] NA NA NA NA NA
```

The only caveats is that with a response variable with 0/1 residuals can take only a limited range of values. For this reason, the raw residuals have usually a strange pattern. Furthermore, the linear model assumptions (e.g., normality of residuals) cannot easily be checked.

Binomial GLM - Raw Residuals



Binomial GLM - Binned (raw) Residuals

Gelman and colleagues [3] proposed a type of residuals called **binned residuals**. The idea is to divide values into a certain number of bins, calculate the average fitted/residual and plot the results. To calculate binned residuals we need to:

- divide the fitted values into n bins. The number is arbitrary but we need each bin to have enough observation to compute a reliable average
- calculate the average fitted value and residual for each bin
- for each bin we can compute the standard error as $SE = \frac{\hat{p}_j(1-p_j)}{n_j}$ where p_j is the average fitted probability and n_j is the number of observation in the bin j
- Then we can plot each bin and the confidence intervals (e.g., as $\pm 2 * SE$) where ~95% of binned residuals should be within the CI if the model is true

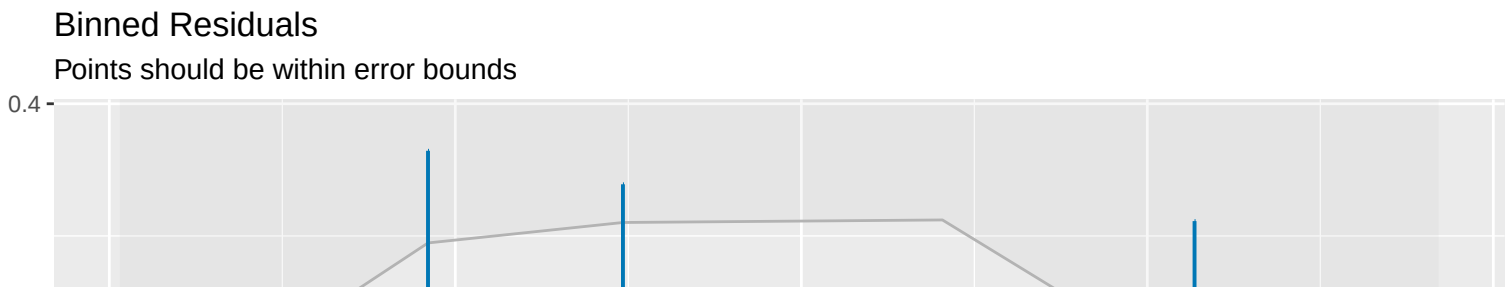
Binomial GLM - Binned (raw) Residuals

We can use the `performance::binned_residuals(model = , n_bins = )` function to automatically create and plot the binned residuals:

```
bres <- performance::binned_residuals(fit, n_bins = 10) # 10 is the default
head(data.frame(bres))
```

```
##          xbar         ybar  n        x.lo        x.hi           se     ci_range
## 1 0.007564827 -0.007564827 10 0.005687965 0.009567447 0.0008341231 0.0004255808
## 2 0.014987600 -0.014987600 10 0.010348263 0.021009376 0.0025855107 0.0013191624
## 3 0.045398938 -0.045398938 10 0.021594415 0.067373399 0.0091720578 0.0046797073
## 4 0.093757831  0.006242169 10 0.069965103 0.130707136 0.2004416089 0.1022680062
## 5 0.230254478  0.069745522 10 0.149751525 0.293573315 0.2946806784 0.1503500476
## 6 0.371099182  0.028900818 10 0.294608733 0.460997399 0.3101562111 0.1582458727
##        CI_low      CI_high group
## 1 -0.00839895 -0.006730704    no
## 2 -0.01757311 -0.012402089    no
## 3 -0.05457100 -0.036226880    no
## 4 -0.19419944  0.206683778   yes
## 5 -0.22493516  0.364426201   yes
## 6 -0.28125539  0.339057029   yes
```

```
plot(bres)
```



Binomial GLM - Pearson residuals

Pearson residuals are raw residuals divided by the standard deviation of each residual. In the case of the binomial GLM:

$$r_i = \frac{y_i - \hat{y}_i}{\sqrt{\hat{p}_i(1-\hat{p}_i)}} \hat{p}_i = logit^{-1}(\hat{y}_i)$$

```
# equivalent to residuals(fit, type = "pearson")
pi <- predict(fit, type = "response")
ri_pearson <- ri / sqrt(pi * (1 - pi))
ri_pearson[1:5]
```

```
##          1          2          3          4          5
## -0.1485632  0.2821712 -0.7120436  0.1567224 -0.0900387
```

Binomial GLM - Standardized Pearson residuals

The standardized pearson residuals are pearson residuals standardized using the hat value of each residual. The hat values are the diagonal of the hat matrix.

$$r_i = \frac{y_i - \hat{y}_i}{\sqrt{sqrt(1 - h_i)}}$$

```
# equivalent to residuals(fit, type = "pearson")
h <- hatvalues(fit) # diagonal of the hat matrix
ri_pearson_std <- ri_pearson / sqrt(1 - h)
ri_pearson_std[1:5]
```

```
##           1           2           3           4           5
## -0.14950930  0.28558190 -0.72044023  0.15788413 -0.09036625
```

An interesting property of **standardized pearson residuals** is that the distribution is approximately normal when the model has a good fitting to the data

Binomial GLM - Deviance residuals

The **deviance residuals** are calculated using the likelihood of the model and as for standard linear model the sum of the squared deviance residuals is the residual deviance used to assess the model goodness of fit.

$$sign(y_i - \hat{y}_i)\sqrt{-2(y_i log(\hat{p}_i) + (1 - y_i) log(1 - \hat{p}_i)}$$

```
# equivalent to residuals(fit, type = "deviance")
ri_deviance <- sign(ri)*sqrt(-2*(dat$y*log(pi) + (1 - dat$y)*log(1 - pi)))
ri_deviance[1:5]
```

```
##          1          2          3          4          5
## -0.2089545  0.3914325 -0.9056765  0.2202958 -0.1270770
```

```
# residual deviance
sum(ri_deviance^2)
```

```
## [1] 69.57113
```

```
fit$deviance
```

```
## [1] 69.57113
```

Binomial GLM - R^2

Compared to the standard linear regression, there are multiple ways to calculate an R^2 like measure for GLMs.

Binomial GLM - Classification accuracy/Error rate

The **error rate** (ER) is defined as the proportion of cases for which the deterministic prediction i.e. guessing $y_i = 1$ if $logit^{-1}(\hat{y}_i) > 0.5$ and guessing \$y_i = \$ if $logit^{-1}(\hat{y}_i) > 0.5$ is wrong. Clearly, $1 - ER$ is the **classification accuracy**.

Binomial GLM - Studentized residuals

1. Beyond the Gaussian distribution
2. Generalized Linear Models
3. Relevant distributions
4. Data simulation *[EXTRA]*
5. Binomial GLM
6. Binomial GLM
7. Binomial GLM - Diagnostic
8. Binomial GLM - Parameter Intepretation

Binomial GLM - Outliers and influential observations

Identification of influential observation and outliers of GLMs is very similar to standard regression models. We will briefly see:

- The hat values
- Cook Distances
- DFBETAs

Binomial GLM - Hat values

The hat values can be understood as the impact of an observed value on the fitted value. As the hat value increase, the observation i can be considered as an influential data point (i.e., high leverage). For a simple linear regression the hat values are calculated as:

$$h_i = \frac{1}{n} + \frac{(X_i - \bar{X})^2}{\sum_{j=1}^{n}(X_i - \bar{X})^2}$$

In simple words, an high h value happen when the observation is distant from the mean. To note, for multiple regression and GLMs the computation is different but the intepretation is the same.

Binomial GLM - Cook Distances

The Cook Distance of an observation i measured the impact of that observation on the overall model fit. If removing the observation i has an high impact, the observation i is likely an influential observation.

plot/formula here

Binomial GLM - DFBETAs

DFBETAs are similar to the Cook Distances but concerning each model parameter separately. Thus the DFBETAs of the observation i on the parameter j is the impact on the estimate of the parameter when removing that observation.

Binomial GLM - Extracting influence measures

In R we can use the `influence.measures()` function to calculate each influence measure explained before[2].

```
infl <- influence.measures(fit)$infmat
head(infl)
```

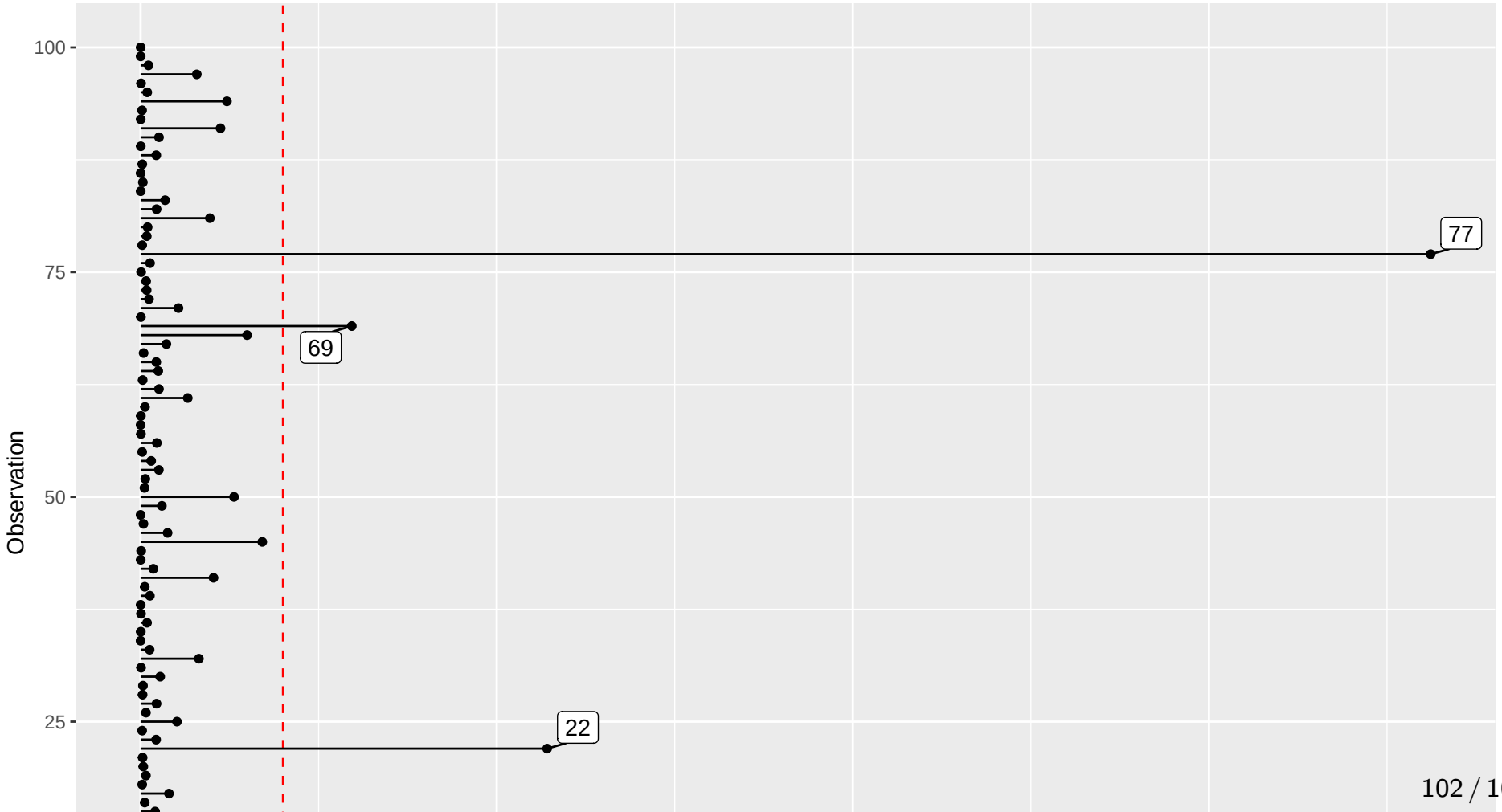
```
##         dfb.1_      dfb.x       dffit    cov.r       cook.d         hat
## 1 -0.02792459 0.02582904 -0.02807601 1.032453 1.428043e-04 0.012615975
## 2 -0.04882965 0.06265696  0.07303366 1.040837 9.917550e-04 0.023743086
## 3 -0.10101996 0.05713430 -0.16767358 1.019868 6.156727e-03 0.023174008
## 4 -0.02463061 0.02954359  0.03197732 1.034447 1.854615e-04 0.014661957
## 5 -0.01285473 0.01219103 -0.01285867 1.027684 2.976067e-05 0.007236127
## 6 -0.01617337 0.01524499 -0.01618929 1.028916 4.723420e-05 0.008580396
```

[2]The function actually computes also other influence measures, see Dunn (2018, section 8.8.3) for other details

Binomial GLM - Plotting influence measures

I wrote the `cook_plot()` function to easily plot the cook distances along with the identification of influential observations:

```
cook_plot(fit)
```



Binomial GLM - Plotting influence measures

I wrote the `dfbeta_plotdfbeta_plots()` function to easily plot the cook distances along with the identification of influential observations:

```
dfbeta_plot(fit)
```