Compulsory exercise 2: Group 13

TMA4315: Generalized linear (mixed) models H2023

Benjamin Sigbjørnsen, Johan Vik Mathisen, and Martinius Theodor Singdahlsen

12 October, 2023

Packages:

```
library(car)
library(ggplot2)
library(GGally)
library(mylm)
library(reshape2)
library(kableExtra)
library(dplyr)
library(mdscore)
library(stats)

sigmoid1 <- function(x) {
    y <- 1/(1 + exp(-x))
    return(y)
}

set.seed(42)
```

Part 1)

For this problem we are going to look at the probability of successfully reaching the summit of the worlds 118 highest mountains. We will use a data set that gives us the predictors topographic prominence and height of the mountain. The summary of the data can be seen below.

```
filepath <- "https://www.math.ntnu.no/emner/TMA4315/2018h/mountains"
mount <- read.table(file = filepath, header = TRUE, col.names = c("height", "prominence",
    "fail", "success"))
summary(mount)
```

```
##      height       prominence        fail          success
##  Min.   :7200   Min.   : 217   Min.   : 0.00   Min.   : 3.00
##  1st Qu.:7312   1st Qu.: 703   1st Qu.: 1.00   1st Qu.: 4.00
##  Median :7462   Median :1300   Median : 3.00   Median :10.00
##  Mean   :7568   Mean   :1562   Mean   : 7.46   Mean   :13.44
##  3rd Qu.:7782   3rd Qu.:2160   3rd Qu.:11.00   3rd Qu.:20.00
##  Max.   :8611   Max.   :4608   Max.   :67.00   Max.   :32.00
```

The data set will be used to model the probability of successfully reaching the summit with a logistic regression.

Thus we assume the following,

$$Y_i \sim \text{Binomial}(n_i, \pi_i) \quad \text{for} \quad i = 1, 2, ..., 113, \tag{1}$$

$$\eta_i = x_i^T \beta, \tag{2}$$

$$\eta_i = \ln(\frac{\pi_i}{1-\pi_i}). \tag{3}$$

Here x_i is the amount of climbers succesfully managing to climb mountain i, and n_i the total amount of climbers that tried to climb mountain i. Further more x_i is a vector of length p, meaning we have $p-1$ predictors.

a)

We will now derive the log likelihood function from the assumptions of the model. In the following calculation, $N = (n_1, ..., n_{113})$. Starting of with the likelihood we obtain,

$$L(x_1, ..., x_{113}, N|\pi_1, ..., \pi_{113}) = \prod_{i=1}^{113} \binom{n_i}{x_i} \pi_i^{x_i} (1-\pi_i)^{1-x_i}. \tag{4}$$

Now to get the log likelihood we simply take the log of the likelihood that we just obtained,

$$\begin{aligned} l(x_1, ..., x_{113}, N|\beta) &= \ln(\prod_{i=1}^{113} \binom{n_i}{x_i} \pi_i^{x_i} (1-\pi_i)^{1-x_i}) && (5)\\ &= \sum_{i=1}^{113} \ln(\binom{n_i}{x_i}) + x_i \ln(\pi_i) + (n_i - x_i)\ln(1-\pi_i) && (6)\\ &= \sum_{i=1}^{113} x_i \ln(\pi_i) - x_i \ln(1-\pi_i) + n_i \ln(1-\pi_i) + \ln(\binom{n_i}{x_i}) && (7)\\ &= \sum_{i=1}^{113} x_i \eta_i + n_i \ln(1-\pi_i) + \ln(\binom{n_i}{x_i}). && (8) \end{aligned}$$

If we maximize this function with regards to β we will find our estimates for β. This can be achieved with for example gradient assent or the Fisher scoring algorithm. The Fisher scoring algorithm is a Newton method in multiple dimensions. The iteration can be described with the following formula,

$$\beta^{j+1} = \beta^j + F^{-1}(\beta^j) s(\beta^j). \tag{9}$$

Here $F(\beta^j)$ is the expected Fisher information and $s(\beta^j)$ is the score function. Since we are looking for the zeros of the score function and the expected fisher information equals the observed fisher information in the binomial case. We see that this is a Newton method.

b)

Fitting a logistic regression model to the data with the function `glm()` we obtain the following:

```
binary_regression_1 <- glm(cbind(success, fail) ~ height + prominence, data = mount,
    family = "binomial")
summary(binary_regression_1)
```

```
##
## Call:
## glm(formula = cbind(success, fail) ~ height + prominence, family = "binomial",
##     data = mount)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -6.2886  -0.8086   0.6893   1.4226   3.7456
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.369e+01  1.064e+00  12.861  < 2e-16 ***
## height      -1.635e-03  1.420e-04 -11.521  < 2e-16 ***
## prominence  -1.740e-04  4.554e-05  -3.821 0.000133 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 715.29  on 112  degrees of freedom
## Residual deviance: 414.68  on 110  degrees of freedom
## AIC: 686.03
##
## Number of Fisher Scoring iterations: 4
```

To interpret the coefficients of the data i will express the change of odds given that either the prominence or the height is a meter higher. Thus we can see how much changing each explanatory variable will change the odds. This is given by the function,

$$\frac{P(Y_i = 1|x_{ji} + 1)}{P(Y_i = 0|x_{ji} + 1)} = \frac{P(Y_i = 1|x_i)}{P(Y_i = 0|x_i)} \exp(\beta_i), \tag{10}$$

where $\exp(\beta_i)$ will be reported. This can be seen in the table below for each β_i.

```
df_of_eb <- data.frame(`exp(coef)` = exp(binary_regression_1$coefficients))
print(df_of_eb)
```

```
##                exp.coef.
## (Intercept) 8.783892e+05
## height      9.983659e-01
## prominence  9.998260e-01
```

We can see from the table that the coefficient `exp(height)` is the smallest compared to `exp(prominence)`. Thus per meter it reduces the odds the most. This means that an increase in a meter of height in the mountain makes it less probable to accomplish a successful summit rather than an increase in prominence. The `exp(intercept)` does not have a clear meaning as no mountain is zero meters tall. However it is the maximal odds you can accomplish with the model.

To test the significance of the whole fit we will preform the Wald test. This will be done by using the function `wald.test()` from the library `mdscore`.

```
wald.test(model = binary_regression_1, terms = 3)
```

```
## $W
## [1] 14.59851
##
```

```
## $pvalue
## [1] 0.00013302
##
## attr(,"class")
## [1] "wald.test"
```

Running the function we get a p-value of $1.33 \cdot 10^{-4}$, which is smaller than the chosen significance level of 95%. Thus the model with all the coefficients is preferred.

Further more we would like to create a confidence interval for $\exp(\beta_{height})$. Let the confidence interval be denoted by, $(\exp(\beta^L), \exp(\beta^H))$. This can be accomplished by first using the asymptotically normality of the parameters then transforming by $f(x) = \exp(x)$. We know for the output of the model that the devience of β_{height}, is $1.420 \cdot 10^{-04}$. We thus obtain the confidence interval,

$$(\hat{\beta}^L, \hat{\beta}^H) = (\beta_{\text{height}} - 1.96 \cdot \sigma_{\text{height}}, \beta_{\text{height}} + 1.96 \cdot \sigma_{\text{height}}) \tag{11}$$
$$= (-0.001913, -0.001357). \tag{12}$$

Since $\exp(x)$ is a monotonic increasing function, the confidence interval for $\exp(\beta_{\text{height}})$ becomes,

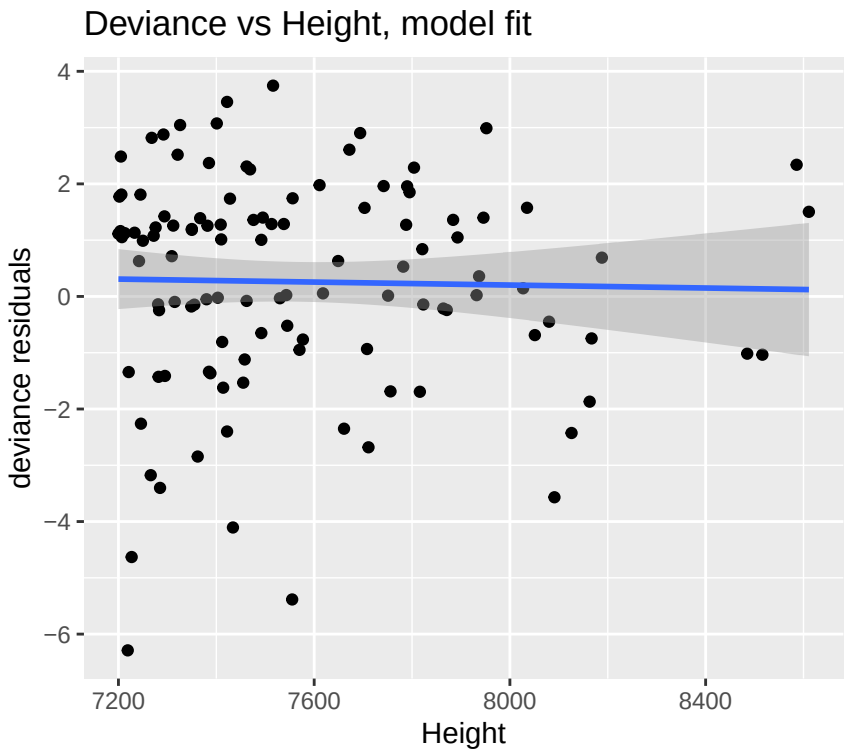
$$(\exp(\hat{\beta}^L), \exp(\hat{\beta}^H)) = (\exp(-0.001913), \exp(-0.001357)) \tag{13}$$
$$= (0.9981, 0.9986). \tag{14}$$

As we can see the confidence interval is quite small. However say that a mountain is 7500 meters tall. Then we will not multiply by $\exp(\beta_{\text{height}})$ however with $\exp(7500\beta_{\text{height}})$. This will result in a larger confidence interval.

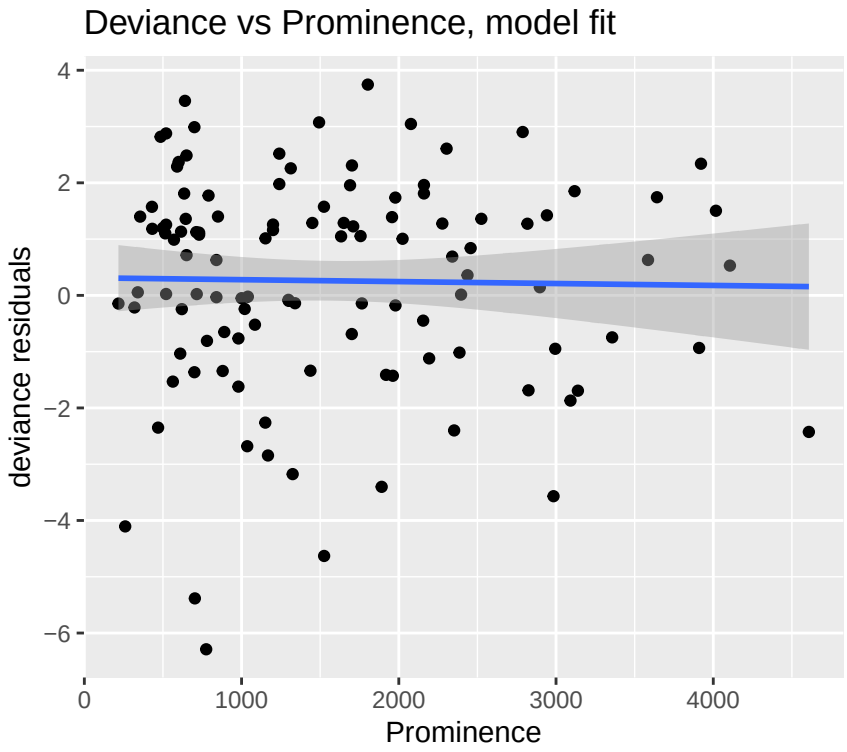
c)

Now after checking the significance of the model we would like to assess the model fit. This is to see if there are indications of any assumptions being violated. Plotting the deviance residuals against both covariate's on two different plots, we get the plots bellow.

```
deviance <- residuals.glm(binary_regression_1, type = "deviance")
df_plot_dev_height <- data.frame(Prominence = mount$prominence, Height = mount$height,
    Deviance = deviance)

ggplot(data = df_plot_dev_height, aes(x = Height, y = Deviance)) + geom_point() +
    ggtitle("Deviance vs Height, model fit") + geom_smooth(method = "glm", formula = y ~
    x) + ylab("deviance residuals")
```



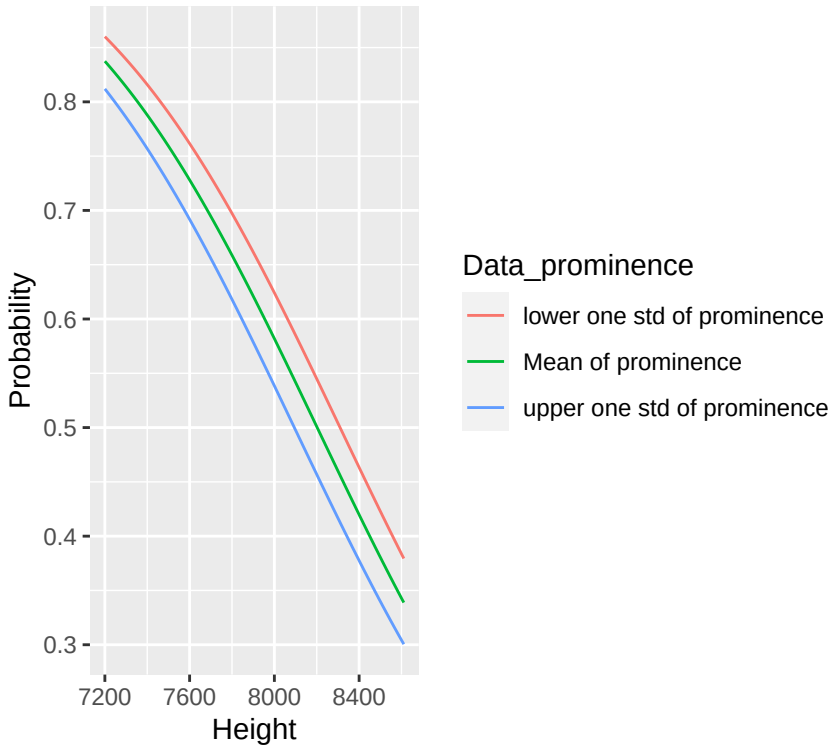

```
ggplot(data = df_plot_dev_height, aes(x = Prominence, y = Deviance)) + geom_point() +
    ggtitle("Deviance vs Prominence, model fit") + geom_smooth(method = "glm", formula = y ~
    x) + ylab("deviance residuals")
```



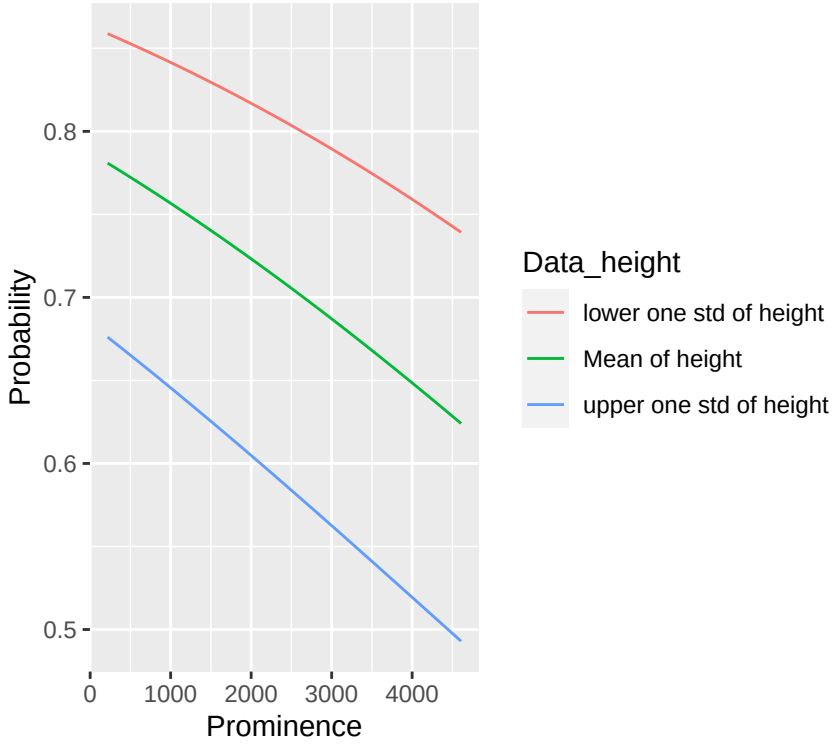
We can see that there are some outliers in both plots that have a deviance of approximately -6. However the plot is mostly centered around zero wich we interpret as the model assumptions holding.

To get a visualization of the probability of successful summit of a mountain, plots of the probability of successful summit is plotted on the y axis and the value of a covariate on the x. Note that this produces two plots as we have two covariates. If the the covariate height is a function of the probability, the mean and one standard deviance of prominence of the data is chosen to be constant. In the corresponding other situation where prominence is a function of the probability, the mean and one standard deviation of height is chosen.

```
sd_prom <- sqrt(var(mount$prominence))
mean_prom <- mean(mount$prominence)
range_height <- range(mount$height)
x_height_var <- seq(from = range_height[1], to = range_height[2], by = 1)
x_01 <- rep(1, length(x_height_var))
x_prom_H_sd <- rep(mean_prom + sd_prom, length(x_height_var))
x_prom_mean <- rep(mean_prom, length(x_height_var))
x_prom_L_sd <- rep(mean_prom - sd_prom, length(x_height_var))

plot_height_M1 <- matrix(c(x_01, x_height_var, x_prom_H_sd), ncol = 3, byrow = FALSE)
plot_height_eta1 <- plot_height_M1 %*% binary_regression_1$coefficients
prob_height_var1 <- sigmoid1(plot_height_eta1)

plot_height_M2 <- matrix(c(x_01, x_height_var, x_prom_mean), ncol = 3, byrow = FALSE)
plot_height_eta2 <- plot_height_M2 %*% binary_regression_1$coefficients
prob_height_var2 <- sigmoid1(plot_height_eta2)

plot_height_M3 <- matrix(c(x_01, x_height_var, x_prom_L_sd), ncol = 3, byrow = FALSE)
plot_height_eta3 <- plot_height_M3 %*% binary_regression_1$coefficients
prob_height_var3 <- sigmoid1(plot_height_eta3)

n = length(prob_height_var1)
df = data.frame(Height = c(x_height_var, x_height_var, x_height_var), Probability = c(prob_height_var1,
    prob_height_var2, prob_height_var3), Data_prominence = c(rep("upper one std of prominence",
    n), rep("Mean of prominence", n), rep("lower one std of prominence", n)))

ggplot(df, aes(Height, Probability, col = Data_prominence)) + geom_line()
```




```
sd_height <- sqrt(var(mount$height))
mean_height <- mean(mount$height)
range_prom <- range(mount$prominence)
x_prom_var <- seq(from = range_prom[1], to = range_prom[2], by = 1)
x_02 <- rep(1, length(x_prom_var))
x_height_H_sd <- rep(mean_height + sd_height, length(x_prom_var))
x_height <- rep(mean_height, length(x_prom_var))
x_height_L_sd <- rep(mean_height - sd_height, length(x_prom_var))

plot_prom_M1 <- matrix(c(x_02, x_height_H_sd, x_prom_var), ncol = 3, byrow = FALSE)
plot_prom_eta1 <- plot_prom_M1 %*% binary_regression_1$coefficients
prob_prom_var1 <- sigmoid1(plot_prom_eta1)

plot_prom_M2 <- matrix(c(x_02, x_height, x_prom_var), ncol = 3, byrow = FALSE)
plot_prom_eta2 <- plot_prom_M2 %*% binary_regression_1$coefficients
prob_prom_var2 <- sigmoid1(plot_prom_eta2)

plot_prom_M3 <- matrix(c(x_02, x_height_L_sd, x_prom_var), ncol = 3, byrow = FALSE)
plot_prom_eta3 <- plot_prom_M3 %*% binary_regression_1$coefficients
prob_prom_var3 <- sigmoid1(plot_prom_eta3)

n = length(prob_prom_var1)
df = data.frame(Prominence = c(x_prom_var, x_prom_var, x_prom_var), Probability = c(prob_prom_var1,
    prob_prom_var2, prob_prom_var3), Data_height = c(rep("upper one std of height",
    n), rep("Mean of height", n), rep("lower one std of height", n)))
ggplot(df, aes(Prominence, Probability, col = Data_height)) + geom_line()
```



Looking at both plots we can see that the covariate height does inn fact influence the probability of acceding the mountain the most. This is reflected by the range of the y-axis in the plot of probability vs height and the differences of the functions plotted in probability vs prominence.

d)

We will now find the probability of ascending mount Everest and Chogolisa by the generalized linear model we fitted. Mount Everest has a height and prominence of bout 8848 meters while Chogolisa has a height of 7665 meters and a prominence of 1624 meters.

```
df.pred_mountain <- data.frame(height = c(8848, 7665), prominence = c(8848, 1624))
predict.glm(binary_regression_1, newdata = df.pred_mountain, type = "response")
```

```
##          1          2
## 0.08917783 0.70429238
```

This gives us the probabilities of acceding mount Everest of 0.089 and the probability of acceding Chogolisa of 0.704.

Now we will construct a confidence interval. To construct a 95% confidence interval we will first calculate the confidence interval for the linear predictor. We know that β is asymptotically normally distributed, $\hat{\beta} \approx N_3(\beta, F^{-1}(\hat{\beta}))$. In our case β is multinormal with dimensions (3×1). If we let, $X = (1, x_{\text{height}}, x_{\text{prominence}})$ (3×1) represent the data, the confidence interval for our linear predictor becomes,

$$\hat{\eta} = X^t\hat{\beta} \tag{15}$$

$$\Rightarrow \hat{\eta} \approx N_1(X^t\beta, X^tF^{-1}(\hat{\beta})X) \tag{16}$$

$$\Rightarrow (\hat{\eta}^L, \hat{\eta}^H) = \left(\hat{\eta} - z_{a/2}\cdot\sqrt{X^tF^{-1}(\hat{\beta})X}, \hat{\eta} + z_{a/2}\cdot\sqrt{X^tF^{-1}(\hat{\beta})X}\right). \tag{17}$$

```
X_eve <- c(1, 8848, 8848)
X_cho <- c(1, 7665, 1624)
var_eve <- t(X_eve) %*% vcov(binary_regression_1) %*% X_eve
var_cho <- t(X_cho) %*% vcov(binary_regression_1) %*% X_cho
eta_eve <- t(X_eve) %*% binary_regression_1$coefficients
eta_cho <- t(X_cho) %*% binary_regression_1$coefficients

eta_eve_L <- eta_eve - 1.96 * sqrt(var_eve)
eta_eve_H <- eta_eve + 1.96 * sqrt(var_eve)
eta_cho_L <- eta_cho - 1.96 * sqrt(var_cho)
eta_cho_H <- eta_cho + 1.96 * sqrt(var_cho)

confinterval_eta <- c(eta_eve_L, eta_eve_H, eta_cho_L, eta_cho_H)
print(confinterval_eta)
```

```
## [1] -2.8464478 -1.8009826  0.7686328  0.9670120
```

After doing the calculations in R-studios we get the 95% confidence interval for the linear predictor of mount Everest of, $(-2.846, -1.801)$. The corresponding 95% confidence interval for Chogolisa is $(0.769, 0.967)$.

To obtain a 95% confidence interval for the probability, we transform by the standard logistic function,

$$h^{-1}(x) = \frac{1}{1+\exp(-x)}, \tag{18}$$

which is monotonically increasing. The formula for the confidence interval for the probability then becomes,

$$(\hat{\pi}^L, \hat{\pi}^H) = \left(h^{-1}\left(\hat{\eta} - z_{a/2} \cdot \sqrt{X^t F^{-1}(\hat{\beta}) X}\right), h^{-1}\left(\hat{\eta} + z_{a/2} \cdot \sqrt{X^t F^{-1}(\hat{\beta}) X}\right)\right) \tag{19}$$

This is simply done in R by applying the function logit function:

```
confinterval_prob <- sigmoid1(confinterval_eta)
print(confinterval_prob)
```

```
## [1] 0.05486523 0.14173150 0.68322506 0.72452353
```

We then get the confidence interval for the prediction of successfully assenting mount Everest of $(0.0549, 0.142)$. It may be reasonable to estimate the probability of ascending Mount Everest whit the data we have fit the model on. The problem occurs because of mount Everest's prominence is 8848 meters while the data we have made the fit on ranges from 217 meters to 4608 meters. The height of mount Everest is also out of the range of the data. This is also reflected by the large confidence interval relative to Chogolisa's confidence interval. If the model assumptions hold well out of the data range we get a prediction that is reasonable. If the model assumptions do not hold well out of the data range the prediction would be bad. However this we can not know with the information we have on our hand at the moment. It probably would be a bad estimation because of how organised the climb of mount Everest is compared to the mountains in the data set. This would in reality give a higher probability of successful assent. Thus i would argue that it is not reasonable to use this model to estimate the probability of climbing mount Everest.

When it comes to Chogolisa the confidence interval for successfully accenting the mountain is $(0.683, 0.725)$. This is a reasonable estimation of the probability as it is within the data range and has similar probabilities to similar mountains.

Part 2)

In this part we are exploring a data set containing results from all played football matches of the 2023 Eliteserien.

```
# Reading the data
rm(list = ls())
filepath <- "https://www.math.ntnu.no/emner/TMA4315/2023h/eliteserien2023.csv"
eliteserie <- read.csv(file = filepath)

NGames <- table(c(eliteserie$home[!is.na(eliteserie$yh)], eliteserie$away[!is.na(eliteserie$yh)]))
RangeofGames <- range(NGames)
```

a)

To test the assumption of independence, we have to perform the test

H_0 : Independence between the goals made by the home and away teams H_1: Dependence between the goals made by the home and away teams

We test using a Pearson's χ^2 test on the contingency table of the goals

```
# Creating contingency table
contingencyTab = table(eliteserie$yh, eliteserie$ya)
# Testing the independence between the goals made by the home and away teams
chisq.test(contingencyTab, simulate.p.value = TRUE)
```

```
##
##  Pearson's Chi-squared test with simulated p-value (based on 2000
##  replicates)
##
## data:  contingencyTab
## X-squared = 33.527, df = NA, p-value = 0.3848
```

The p-value of the test is 0.38, and we therefore fail to reject our null hypothesis using every reasonable significance level. We can not conclude with dependence between goals made by the home and away teams

b)

The preliminary ranking for the season as of October 1st is presented in table 1.

```
tableFunc = function(eliteserie) {
    # Removing matches that are not played
    eliteserie = eliteserie[!is.na(eliteserie$yh), ]

    # Points home team
    eliteserie$ph = 0
    eliteserie[eliteserie$yh > eliteserie$ya, ]$ph = 3
    eliteserie[eliteserie$yh == eliteserie$ya, ]$ph = 1

    # Points away team
    eliteserie$pa = 1
    eliteserie[eliteserie$ph != eliteserie$pa, ]$pa = 3 - eliteserie[eliteserie$ph !=
        eliteserie$pa, ]$ph

    # Creating table
    table = data.frame(team = c(eliteserie$home[1:8], eliteserie$away[1:8]), points = 0,
```

Table 1: Preliminary ranking for the season as of October 1st

team	points	goal_difference
Viking	51	21
Bodø/Glimt	49	28
Tromsø	48	11
Brann	42	14
Molde	41	24
Lillestrøm	36	7
Sarpsborg 08	34	5
Odd	30	-3
Rosenborg	29	-6
Strømsgodset	27	-3
HamKam	24	-15
Vålerenga	21	-8
Sandefjord Fotball	21	-9
Haugesund	21	-14
Stabæk	17	-16
Aalesund	12	-36

```
        goal_difference = 0)

    # Giving teams points
    for (i in 1:nrow(eliteserie)) {
        teamHome = eliteserie$home[i]
        teamAway = eliteserie$away[i]
        table[table$team == teamHome, ]$points = table[table$team == teamHome, ]$points +
            eliteserie$ph[i]
        table[table$team == teamAway, ]$points = table[table$team == teamAway, ]$points +
            eliteserie$pa[i]
        table[table$team == teamHome, ]$goal_difference = table[table$team == teamHome,
            ]$goal_difference + eliteserie$yh[i] - eliteserie$ya[i]
        table[table$team == teamAway, ]$goal_difference = table[table$team == teamAway,
            ]$goal_difference + eliteserie$ya[i] - eliteserie$yh[i]
    }

    # Sorting by points, and then goal difference
    table = table[order(table$points, table$goal_difference, decreasing = TRUE),
        ]
    return(table)
}
table = tableFunc(eliteserie)
rownames(table) <- 1:nrow(table)
table %>%
    kbl(caption = "Preliminary ranking for the season as of October 1st \\label{tab:prelim_rank}") %>%
    kable_classic(full_width = F, html_font = "Cambria")
```

c)

In this exercise we implement a function estimating the intercept, home advantage and strength parameters for each team in Eliteserien 2023 based on the 192 games played as of October 1.

We build our design matrix X as explained by the hint in the project description.

```
# Removing nans.
eliteserie_clean <- na.omit(eliteserie)
# Extracting nr of games played so far this season.
games_played = dim(eliteserie_clean)[1]
# Extracting team names.
teams = as.array(unique(eliteserie$home))


# Initialize the design matrix X, which is a 2*192 x 18 matrix.  Each row in X
# corresponds to a played game. Once from each side. Therefore 2*games_played
# rows.

# Initialize all zeros matrix
X = matrix(0, nrow = games_played * 2, ncol = 18)

# Set proper feature names
colnames(X) = c("Intercept", "HomeAdvantage", teams)

# First we alter the design matrix for all 'home games'
for (i in 1:games_played) {
    # Intercept
    X[i, 1] = 1

    # HomeAdvantage = 1
    X[i, 2] = 1

    # x_homeTeam = 1
    X[i, match(eliteserie_clean[i, 2], colnames(X))] = 1

    # x_awayTeam = -1
    X[i, match(eliteserie_clean[i, 3], colnames(X))] = -1
}

# Then alter design matrix for all 'away games'
for (i in 1:games_played) {

    # Intercept
    X[games_played + i, 1] = 1

    # Same as above, but inverted:

    # x_home = 0 from initialization.

    # x_homeTeam = -1
    X[games_played + i, match(eliteserie_clean[i, 2], colnames(X))] = -1

    # x_awayTeam is set to 1
    X[games_played + i, match(eliteserie_clean[i, 3], colnames(X))] = 1
}

# Response vector, all goals.
Y = c(eliteserie_clean$yh, eliteserie_clean$ya)
```

```r
Y = matrix(Y, length(Y), 1)


# Removed to make mle solution unique and optim to work.
X <- X[, -which(colnames(X) == "Bodø/Glimt")]
```

For parameter estimation we use maximum likelihood estimation. For a Poisson regression model we know that the MLE does not have a closed form solution, hence we turn to numerics in order to obtain our estimates.

Since our design matrix originaly has linearly dependent columns, the MLE has infinitely many solutions. We do as proposed by the exercise text and force $x_{\text{Bodø/Glimt}}$ to be zero, leaving us with a unique MLE. In our model the strength parameter for Bodø/Glimt will then be the reference value 0.

The log likelihood of a Poisson sample is given by

$$l(\beta) = \sum_{i=1}^{n} \left[y_i \ln(\lambda_i) - \lambda_i - \ln(y_i!) \right].$$

As we use the `R` function `Optim`, and this by default minimizes, we implement the negative log likelihood below.

By including the data-dependent term $\ln(y_i!)$ in our calculations we got extremely accurate results, and therefore choose not to include gradient information to further increas accuracy.

```r
neg_log_likelihood_poisson <- function(beta) {
    eta <- X %*% beta
    lambda <- exp(eta)
    return(-(t(Y) %*% eta - sum(lambda - log(factorial(Y)))))
}
my_poisson <- function(X, Y) {
    "
X: Design matrix
Y: Response (possion distributed)
"
    # Initial value for beta.
    beta_init <- rep(0.5, 17)

    beta <- optim(beta_init, neg_log_likelihood_poisson, method = "BFGS")$par
    result <- data.frame(Names = colnames(X), Estimate = round(beta, 5))
    return(result)

}
result <- my_poisson(X, Y)

# Remove intercept and homeAdvantage.
power_ranking <- result[-(1:2), ]
# Adding Bodø/Glimt back
power_ranking <- rbind(power_ranking, list(Names = "Bodø/Glimt", Estimate = 0))

# Order teams by power parameter.
power_ranking <- power_ranking[order(power_ranking[, 2], decreasing = TRUE), ]
rownames(power_ranking) <- 1:nrow(power_ranking)

# Power rankings
```

Table 2: Power ranking based on Poisson regresson coefficients.

Names	Estimate
Bodø/Glimt	0.00000
Molde	-0.10314
Viking	-0.12260
Brann	-0.24826
Tromsø	-0.26013
Lillestrøm	-0.32865
Sarpsborg 08	-0.34396
Odd	-0.44285
Strømsgodset	-0.47908
Sandefjord Fotball	-0.50867
Rosenborg	-0.51028
Vålerenga	-0.52032
Haugesund	-0.57215
Stabæk	-0.63302
HamKam	-0.63641
Aalesund	-0.91919

```
power_ranking %>%
    kbl(caption = "Power ranking based on Poisson regresson coefficients.\\label{tab:power_rank}") %>%
    kable_classic(full_width = F, html_font = "Cambria")
```

```
# Parameter estimates
result %>%
    kbl(caption = "Estimates for the regressson coefficients.\\label{tab:reg_coeff}") %>%
    kable_classic(full_width = F, html_font = "Cambria")
```

Preliminary ranking vs strength parameter ranking When we compare the ranking based on the estimated strengths presented in table 2 to the preliminary standings in table 1, we observe that the strength parameter follows the goal difference rather than the points. This is not very surprising as our model only takes goal difference into account.

Our results compared to `glm()` From the summary of the `glm()` model below and our estimated regression coefficients presented in table 3, we see that the estimated coefficients are identical. This leaves us confident in the correctness of our implementation.

```
##
## Call:
## glm(formula = Y ~ -1 + X, family = "poisson")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2652  -1.1581  -0.1047   0.5316   2.4022
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## XIntercept          0.16644    0.06849   2.430 0.015091 *
## XHomeAdvantage      0.35856    0.08745   4.100 4.13e-05 ***
## XRosenborg         -0.51028    0.16443  -3.103 0.001914 **
## XAalesund          -0.91919    0.16662  -5.517 3.46e-08 ***
```

Table 3: Estimates for the regressson coefficients.

Names	Estimate
Intercept	0.16644
HomeAdvantage	0.35856
Rosenborg	-0.51028
Aalesund	-0.91919
HamKam	-0.63641
Sarpsborg 08	-0.34396
Stabæk	-0.63302
Tromsø	-0.26013
Brann	-0.24826
Lillestrøm	-0.32865
Viking	-0.12260
Haugesund	-0.57215
Molde	-0.10314
Odd	-0.44285
Sandefjord Fotball	-0.50867
Strømsgodset	-0.47908
Vålerenga	-0.52032

```
## XHamKam             -0.63641    0.16580  -3.838 0.000124 ***
## XSarpsborg 08       -0.34396    0.16839  -2.043 0.041089 *
## XStabæk             -0.63302    0.16909  -3.744 0.000181 ***
## XTromsø             -0.26013    0.16452  -1.581 0.113831
## XBrann              -0.24826    0.16821  -1.476 0.139978
## XLillestrøm         -0.32865    0.16986  -1.935 0.053014 .
## XViking             -0.12260    0.16428  -0.746 0.455518
## XHaugesund          -0.57215    0.16436  -3.481 0.000499 ***
## XMolde              -0.10314    0.16687  -0.618 0.536537
## XOdd                -0.44285    0.16478  -2.688 0.007197 **
## XSandefjord Fotball -0.50867    0.17038  -2.986 0.002830 **
## XStrømsgodset       -0.47908    0.17080  -2.805 0.005032 **
## XVålerenga          -0.52032    0.16594  -3.136 0.001715 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 555.74  on 358  degrees of freedom
## Residual deviance: 397.07  on 341  degrees of freedom
## AIC: 1102
##
## Number of Fisher Scoring iterations: 5
```

d)

In this exercise we estimate the rankings by means of simmulation, based on the estimated strengths found in the previous task. We do this by simmulating the remaining 61 matches 1000 times.

In order to obtain the probabilities of the different outcomes of the remaining matches we observe that team A winning over team B is the same as the score Y_A being greater than the score Y_B. From our model-assumptions these variables are independent of each other. From this we obtain that

$$\begin{aligned} P(A \text{ wins over } B) &= P(A \text{ scores more than } B) \\ &= \sum_{k=1}^{\infty} P(Y_A = k, Y_B < k) \\ &= \sum_{k=1}^{\infty} \sum_{j=1}^{k} P(Y_A = k) P(Y_B = j) \end{aligned}$$

Similarly for draw and loss we have

$$P(A \text{ loses to } B) = \sum_{k=1}^{\infty} \sum_{j=1}^{k} P(Y_B = k) P(Y_A = j),$$

and

$$P(\text{Draw}) = \sum_{k=0}^{\infty} P(Y_A = k) P(Y_B = k).$$

Let A be the home team, then from our model we have that Y_A is Poisson with

$$\ln E(Y_A) = \ln(\lambda_A) = \eta_A = \beta_0 + \beta_{\text{home}} + \beta_A - \beta_B$$

and Y_B is Poisson with

$$\ln E(Y_B) = \ln(\lambda_B) = \eta_B = \beta_0 - \beta_A + \beta_B$$

when A plays against B. From this we can calculate the desired probabilities.

The sums involved when calculating the probabilities converge extremely quickly, even for $\lambda_{\text{Bodø/Glimt}} = 4.238418$, which is the largest of all teams, the summands are of magnitude less than 10^{-8} for $k = 20$. Thus we take the liberty to say that the sums have converged when $k = 20$.

```
predict_season <- function(Unplayed, Estimates) {
    "
  Unplayed: home, away
  Estimates: df of poisson regression estimates with names.

  returns: df with match outcome probabilities. Home, Away, Win, Draw, Loss
  "
    Predict <- data.frame(Home = character(0), Away = character(0), Win = numeric(0),
        Draw = numeric(0), Loss = numeric(0))

    for (i in 1:nrow(Unplayed)) {
        home <- Unplayed$home[i]
        away <- Unplayed$away[i]
        p_w <- prob_win(home, away, Estimates)
        p_d <- prob_draw(home, away, Estimates)
        p_l <- prob_loss(home, away, Estimates)
        data <- list(Home = home, Away = away, Win = p_w, Draw = p_d, Loss = p_l)
        Predict <- rbind(Predict, data)
    }
    return(Predict)
}
```

```r
sim <- function(Predict) {
    "
  Uses Predict df to simulate the remaining season
  "
    outcome <- apply(Predict[, -(1:2)], 1, function(row) {
        return(sample(c("Win", "Draw", "Loss"), 1, prob = row))
    })
    return(outcome)
}
```

```r
predict_to_table <- function(Predict, outcome, current_standings) {
    "
    Updates the standings according to the
    simulation.

    Predict: Home, Away, probabilities,
    outcome: Sim; result of simulation

    current_standings: Team, points
    "
    Predict <- cbind(Predict, Sim = outcome)

    table <- current_standings
    for (i in 1:nrow(Predict)) {
        if (Predict$Sim[i] == "Win") {
            table[table$team == Predict$Home[i], 2] <- table[table$team == Predict$Home[i],
                2] + 3
        } else if (Predict$Sim[i] == "Loss") {
            table[table$team == Predict$Away[i], 2] <- table[table$team == Predict$Away[i],
                2] + 3
        } else {
            # Draw
            table[table$team == Predict$Home[i], 2] <- table[table$team == Predict$Home[i],
                2] + 1
            table[table$team == Predict$Away[i], 2] <- table[table$team == Predict$Away[i],
                2] + 1
        }
    }
    table <- table[order(table$points, decreasing = TRUE), ]
    return(table)
}
```

```r
sim_season_finish <- function(number_of_sims = 1000, current_standing, Unplayed,
    Estimate) {
    "
Combines the above functions to simulate the season number_of_sims times.
"
    Predict <- predict_season(Unplayed, Estimate)
    outcome <- sim(Predict)
    table <- predict_to_table(Predict, outcome, current_standing)
    for (i in 1:number_of_sims) {
        outcome <- sim(Predict)
        table_1 <- predict_to_table(Predict, outcome, current_standing)
        table <- inner_join(table, table_1, by = "team")
```

```
    }
    # Renames the columns such that results from simulation i is in column 'i'
    colnames(table)[2:ncol(table)] <- 1:(ncol(table) - 1)
    as.data.frame(table)
    return(table)
}
```

```
get_summary_data <- function(sim_result) {
    "
  Counts the number of times each team finishes at the top of the table.
  Calculates mean points obtained for each team.
  Calculates standard deviation of points for each team.
  "
    N <- ncol(sim_result) - 1  #not including team column. This is the number of sims.
    series_win <- data.frame(`SeasonsWon(%)` = rep(0, length(sim_result$team)))
    rownames(series_win) <- sim_result$team
    for (i in 2:ncol(res)) {
        series_win[res[which.max(sim_result[, i]), 1], ] <- series_win[sim_result[which.max(res[,
            i]), 1], ] + 1
    }
    mean <- apply(sim_result[, -1], 1, function(row) {
        return(mean(row))
    })
    std.d <- apply(sim_result[, -1], 1, function(row) {
        return(sd(row))
    })
    summary_data <- list(Mean = mean, Std.dev = std.d, SeasonsWon = series_win/N)
    summary_data <- as.data.frame(summary_data)

    return(summary_data[order(-summary_data$Mean), ])
}
```

In table 4 we see the average number of points and the associated standard deviations, together with the proportion of seasons won by each of the teams in the 1000 simmulated season endings. The low standard deviation stands to us out as the least true to reality for a game of this nature. In figure 2 and 3 we see the empirical point distributions for each of the teams, both jointly and separately.

Table 4: Summary data from 1000 end of season simulations.

	Mean	Std.dev	SeasonsWon...
Bodø/Glimt	66.61139	3.015598	0.5814186
Viking	66.29071	3.362202	0.3986014
Tromsø	59.59940	3.049649	0.0159840
Molde	55.67433	3.463788	0.0029970
Brann	54.14386	3.696118	0.0009990
Lillestrøm	48.14386	3.516431	0.0000000
Sarpsborg 08	44.79321	3.061404	0.0000000
Odd	39.61139	3.110921	0.0000000
Rosenborg	36.83516	3.284175	0.0000000
Strømsgodset	36.37363	3.424363	0.0000000
Sandefjord Fotball	32.36663	3.594224	0.0000000
HamKam	31.48551	3.368982	0.0000000
Vålerenga	30.63137	3.485825	0.0000000
Haugesund	30.29271	3.217955	0.0000000
Stabæk	25.06294	3.232187	0.0000000
Aalesund	15.20879	2.380622	0.0000000

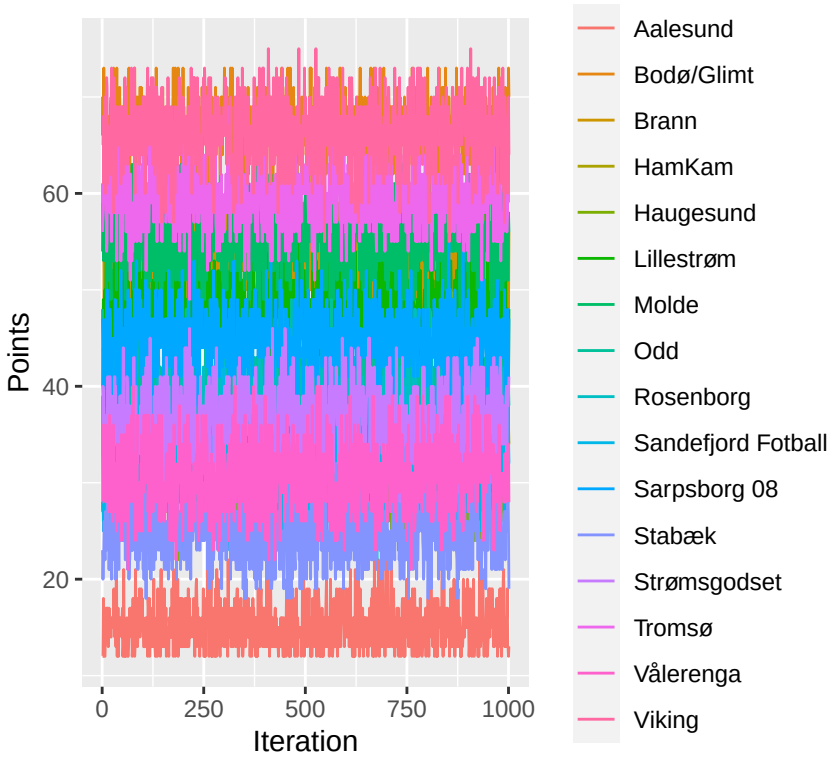


Figure 1: Simulated points at the end of the season, given the standings as of Oct. 1.

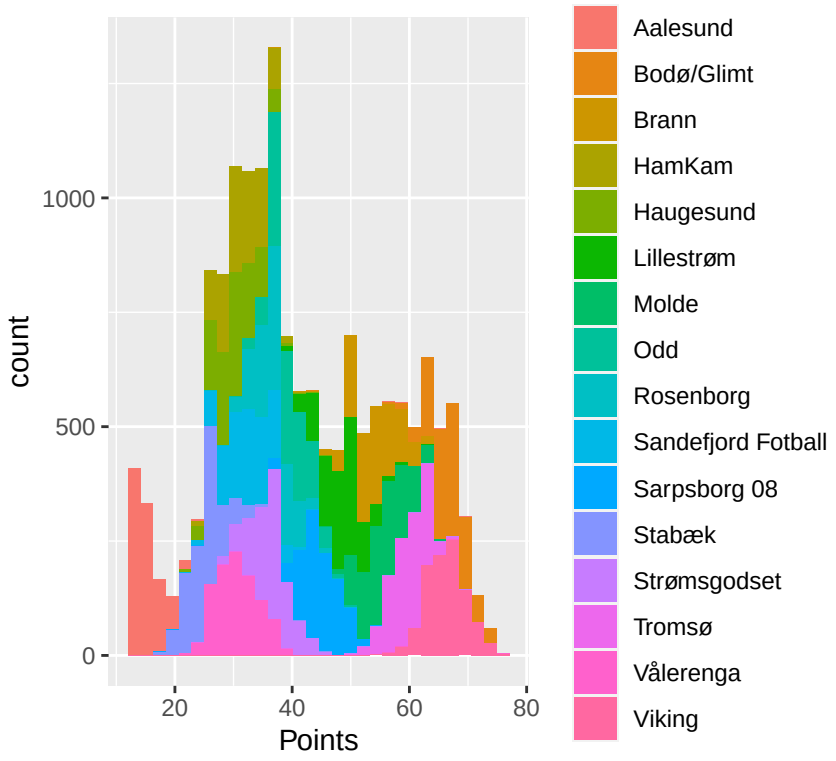


Figure 2: End of season point distribution after 1000 simulations, given the standings as of Oct. 1.

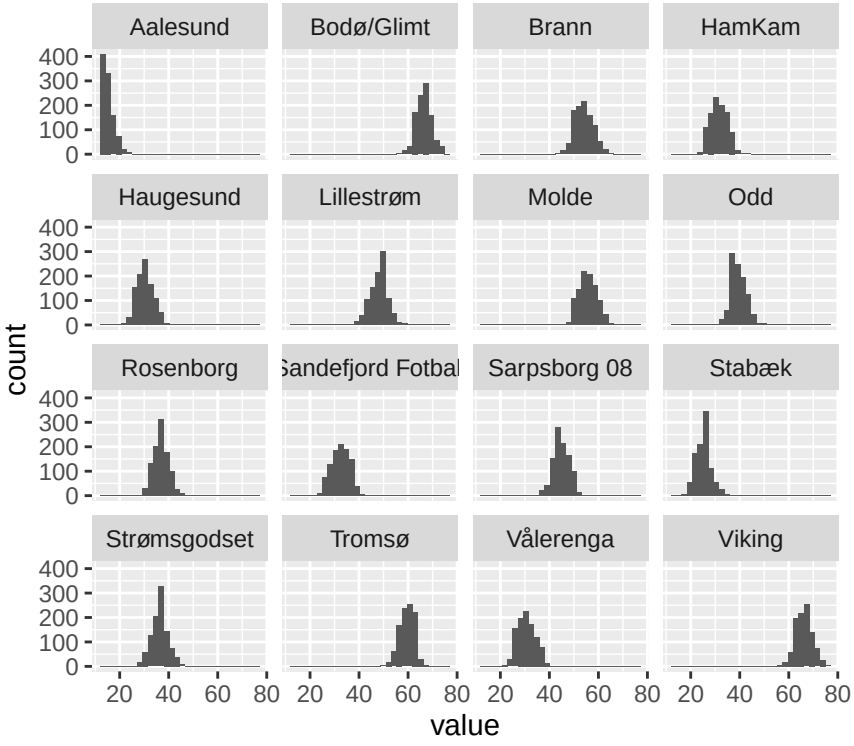


Figure 3: End of season point distribution after 1000 simulations for the individual teams, given the standings as of Oct. 1.