TMA4315: Compulsory Exercise 2

Group 12: Tiago Alexandre Alcobia Pereira

11.10.2024

Part 1: Logistic regression

(a)

We start by writing the likelihood function for a binomial model:

$$L(\beta) = \prod_{i=1}^{n} f(y_i;\beta) = \prod_{i=1}^{n} \binom{n_i}{y_i} \pi_i^{y_i} (1-\pi_i)^{n_i-y_i},$$

where:

- n_i is the number of people trying to climb,
- y_i is the number of people that succeed at climbing, and
- π_i is the probability of success for the i-th observation.

Log-Likelihood

Taking the logarithm of the likelihood function gives us the log-likelihood:

$$l(\beta) = \log L(\beta) = \sum_{i=1}^{n} \left[\log \binom{n_i}{y_i} + y_i \log(\pi_i) + (n_i - y_i)\log(1-\pi_i) \right].$$

Next, we express π_i in terms of β through the logistic function:

$$\pi_i = \frac{e^{\beta x_i^T}}{1+e^{\beta x_i^T}},$$

Substituting this into the log-likelihood, we obtain:

$$l(\beta) = \sum_{i=1}^{n} \left[\log \binom{n_i}{y_i} + y_i \log\left(\frac{e^{\beta x_i^T}}{1+e^{\beta x_i^T}} \right) + (n_i - y_i)\log\left(\frac{1}{1+e^{\beta x_i^T}} \right) \right].$$

Score Function and Maximum Likelihood Estimation

To find the maximum likelihood estimate (MLE) of β, we differentiate the log-likelihood with respect to β, resulting in the score function:

$$S(\beta) = \frac{\partial l(\beta)}{\partial \beta}.$$

Setting the score function equal to zero gives the system of equations for finding $\hat{\beta}$:

$$S(\beta) = 0.$$

This system is typically solved using iterative methods, such as **Newton-Raphson** or **Fisher Scoring**. In each iteration, the estimate of β is updated as:

$$\beta^{(t+1)} = \beta^{(t)} + F(\beta^{(t)})^{-1} S(\beta^{(t)}),$$

where $F(\beta)$ is the Fisher information matrix, which can be obtained by either:

- Taking the expectation of the second derivative of the log-likelihood, or
- Computing the covariance matrix of the score function.

Mathematically, the Fisher information matrix is given by:

$$F(\beta) = E\left(-\frac{\partial l(\beta)}{\partial \beta\, \partial \beta^T}\right) = \text{Cov}(S(\beta))$$

(b)

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

```
##
## Call:
## glm(formula = cbind(success, fail) ~ height + prominence, family = "binomial",
##     data = mount)
##
## 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
```

The interpretation of the model parameters can be understood in terms of *odds ratios*. The log-odds model is defined as:

$$\eta_i = \log\left(\frac{\pi_i}{1-\pi_i}\right)$$

This leads to the relationship between odds and the predictors:

$$\frac{\pi_i}{1-\pi_i} = e^{\beta_0} \times e^{\beta_{\text{height}} \cdot x_{i,\text{height}}} \times e^{\beta_{\text{prominence}} \cdot x_{i,\text{prominence}}}$$

If $x_{i,\text{height}}$ increases by 1 unit, the odds will change by a multiplicative factor of $e^{\beta_{\text{height}}}$. The updated odds are given by:

$$\text{odds}_{\text{new}} = \text{odds}_{\text{old}} \times e^{\beta_{\text{height}}}$$

Also, if $x_{i,\text{prominence}}$ increases by 1 unit, the odds will change by $e^{\beta_{\text{prominence}}}$. Now we calculate and from the code below we notice negative coefficients. The negative coefficients imply that the exponential transformation of these values gives results below 1, implying a decline in the odds. Therefore, as the height and prominence of a mountain increase, the likelihood of a successful ascent decreases. In addition, the influence of height on this likelihood is more significant than of the prominence.

```r
oddsHeight <- exp(logitmodel$coefficients["height"])
oddsProminence <- exp(logitmodel$coefficients["prominence"])
oddsHeight
```

```
##    height
## 0.9983659
```

```r
oddsProminence
```

```
## prominence
##   0.999826
```

Now we test the importance of the predictors by performing Wald tests, where the null hypothesis says that the coefficient is zero. Under the assumption of normality, the Wald statistic is:

$$\frac{\hat{\beta}_i}{\sigma(\hat{\beta}_i)} \sim N(0,1)$$

```r
sd <- sqrt(diag(vcov(logitmodel)))
p <- numeric(length(logitmodel$coefficients))
for (i in seq_along(logitmodel$coefficients)) {
  p[i] <- 2 * pnorm(abs(logitmodel$coefficients[i] / sd[i]), lower.tail = FALSE)
}
names(p) <- names(logitmodel$coefficients)
p
```

```
##  (Intercept)       height   prominence
## 7.477210e-38 1.031229e-30 1.330200e-04
```

From this code we get very small p-values, meaning our parameters are significant for every normal significance level.

Now we find the confidence intervals for both the coefficients and the odds ratios.

Since the estimated coefficients are asymptotically normally distributed, we can calculate 95% confidence intervals.Here $z_{0.025}$ will be the 97.5th percentile of the standard normal distribution.

$$[\hat{\beta}_i - z_{0.025}\sigma(\hat{\beta}_i), \hat{\beta}_i + z_{0.025}\sigma(\hat{\beta}_i)]$$

```
zVal <- qnorm(0.975)
coeff <- logitmodel$coefficients
sd <- sqrt(diag(vcov(logitmodel)))
confidenceIntervals <- list()
for (i in seq_along(coeff)) {
  ciLow <- coeff[i] - zVal * sd[i]
  ciUp <- coeff[i] + zVal * sd[i]
confidenceIntervals[[names(coeff)[i]]] <- c(lower = ciLow, upper = ciUp)
}
confidenceIntervals
```

```
## $`(Intercept)`
## lower.(Intercept) upper.(Intercept)
##          11.60015          15.77154
##
## $height
## lower.height upper.height
## -0.001913631 -0.001357205
##
## $prominence
## lower.prominence upper.prominence
##    -2.632283e-04    -8.473354e-05
```

To obtain confidence intervals for the odds ratios, we simply take the exponent of the limits of the confidence intervals for the coefficients.

```
expConfidence <- list()
for (i in 2:length(coeff)) {
  expCiLow <- exp(confidenceIntervals[[names(coeff)[i]]][1])
  expCiUp <- exp(confidenceIntervals[[names(coeff)[i]]][2])
  expConfidence[[names(coeff)[i]]] <- c(lower = expCiLow, upper = expCiUp)
}
expConfidence
```

```
## $height
## lower.lower.height upper.upper.height
##          0.9980882          0.9986437
##
## $prominence
## lower.lower.prominence upper.upper.prominence
##              0.9997368              0.9999153
```

For the first piece of code, the confidence intervals for β_{height} and $\beta_{\text{prominence}}$ do not extend above 1. This allows us to say, with 95% confidence, that both height and prominence negatively impact the odds of success. For the last piece of code, the confidence intervals for both parameters fall below zero, suggesting that we can be 95% confident that each parameter has a negative effect on the odds.

(c)

The deviance for a model can be expressed as:

$$D = -2\left(l(\hat{\beta}) - l(\tilde{\beta})\right) = -2\left(l(\hat{\pi}) - l(\tilde{\pi})\right),$$

The value $\tilde{\pi}$ corresponds to the probabilities obtained from the fully saturated model, where $\tilde{\pi}_i = \frac{y_i}{n_i}$, ensuring that the predicted number of successes matches the observed number for each mountain. The deviance for an individual mountain i is given by:

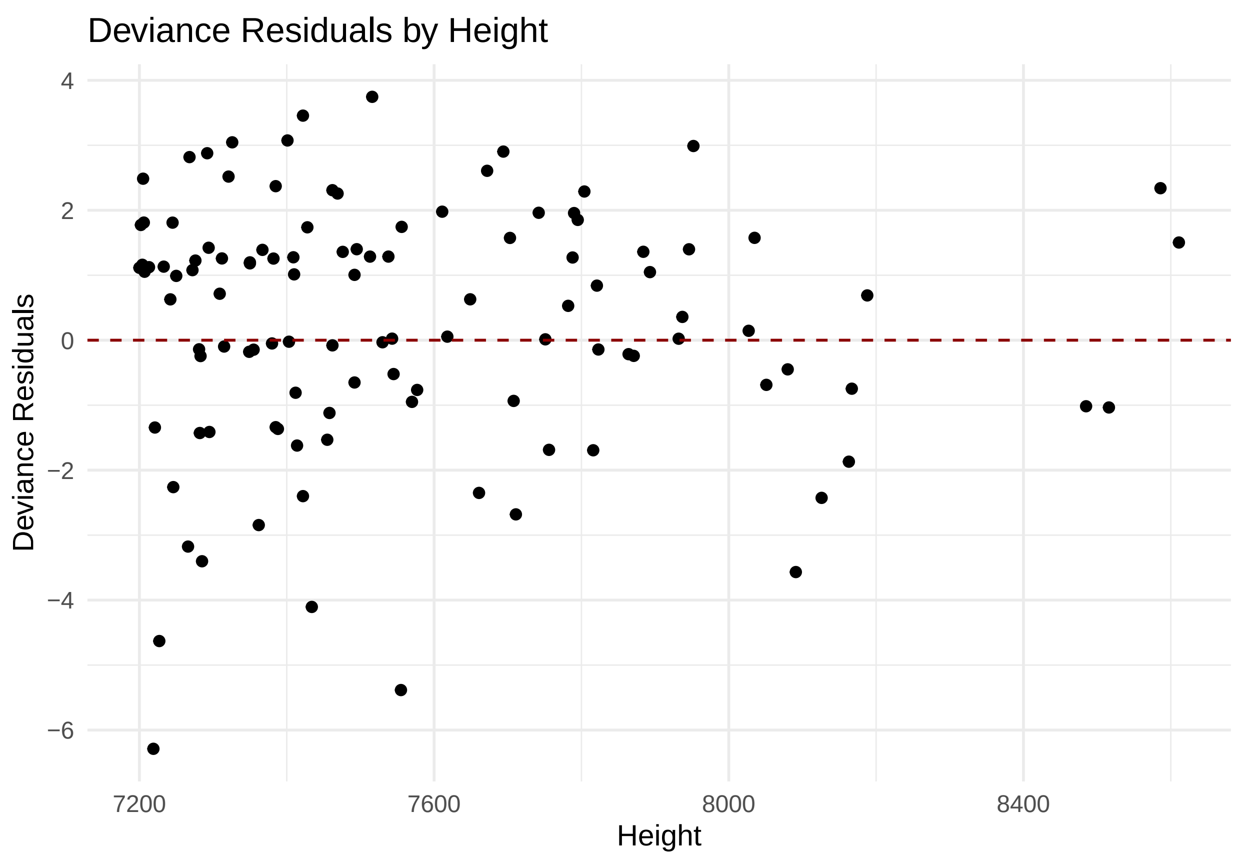
$$D_i = \operatorname{sign}(y_i - \hat{y}_i) \cdot (-2)\left(l(\hat{\pi}_i) - l(\tilde{\pi}_i)\right)$$

This expands to:

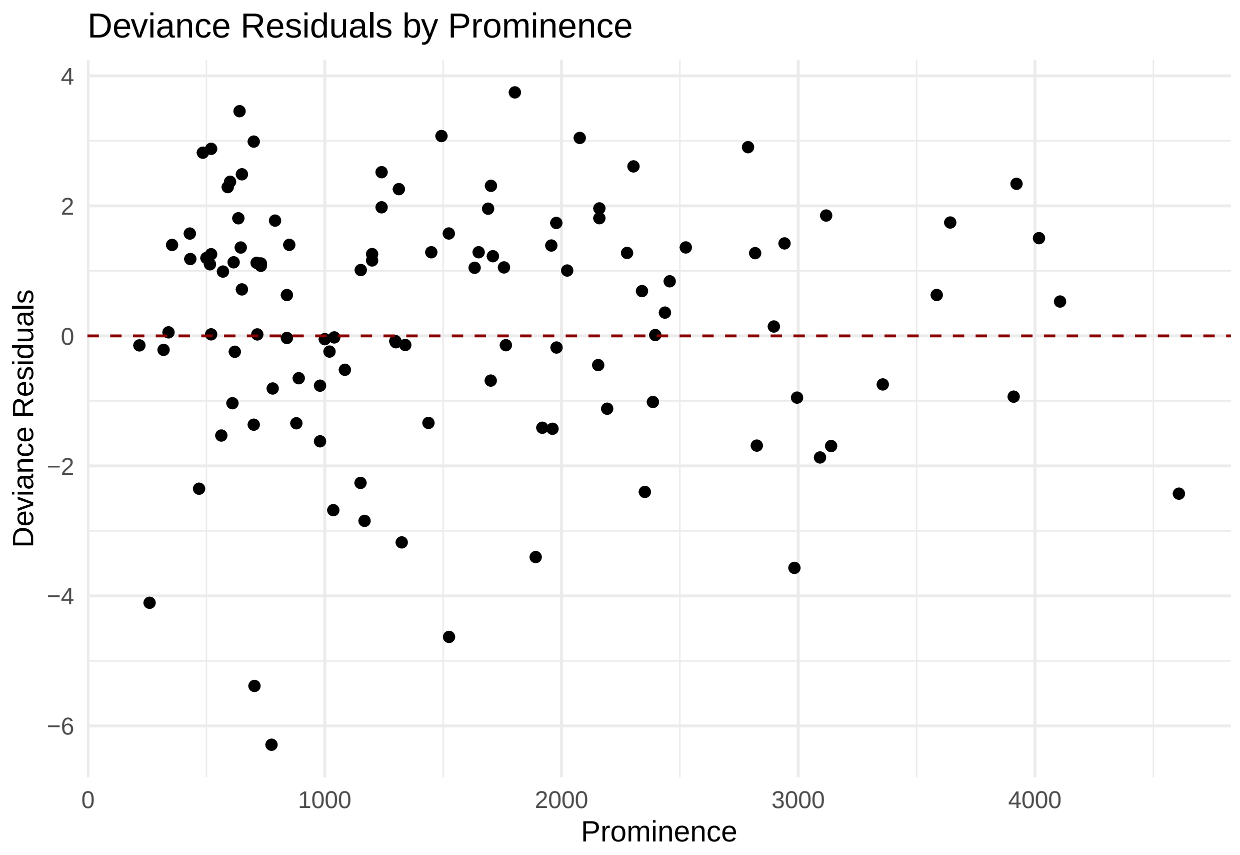
$$D_i = \operatorname{sign}(y_i - \hat{\pi}_i n_i) \cdot (-2)\left(n_i\left(\log(\hat{\pi}_i) - \log\left(\frac{y_i}{n_i}\right)\right) + (n_i - y_i)\left(\log(1-\hat{\pi}_i) - \log\left(1 - \frac{y_i}{n_i}\right)\right)\right).$$

Now we compute the deviance for each mountain and create plots to see the deviance in relation to both height and prominence.

```r
devianceRes <- residuals(logitmodel, type = "deviance")

plotData <- data.frame(
  height_var = mount$height,
  prominence_var = mount$prominence,
  dev_residual = devianceRes
)

library(ggplot2)
ggplot(plotData, aes(x = height_var, y = dev_residual)) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed", color = "darkred") +
  ggtitle("Deviance Residuals by Height") +
  ylab("Deviance Residuals") +
  xlab("Height") +
  theme_minimal()
```



```
ggplot(plotData, aes(x = prominence_var, y = dev_residual)) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed", color = "darkred") +
  ggtitle("Deviance Residuals by Prominence") +
  ylab("Deviance Residuals") +
  xlab("Prominence") +
  theme_minimal()
```



The residuals are mostly scattered around zero for both shorter and taller mountains. However, the model seems to underestimate the chances of success on the smallest mountains, as there are a few large residuals there. For slightly taller mountains, the residuals are generally smaller and closer to zero. Due to the "easier" mountains, the model often ends up overestimating the probability of success on more typical mountains, where negative residuals are more common than positive ones.

A similar behavior can be seen with prominence. The residuals are fairly random around zero, aside from a few exceptions at lower prominence, where the model underpredicts success rates. For mountains with higher prominence, the residuals are generally smaller. Once again, there tend to be more large negative residuals than positive ones.

Next, we will plot the estimated success probabilities in relation to height and prominence.

$$\hat{\pi}_i = \frac{e^{\hat{\beta} x_i^T}}{1 + e^{\hat{\beta} x_i^T}}.$$

We compute this probability across a range of heights and prominences that span the observed values in the dataset, and plot.

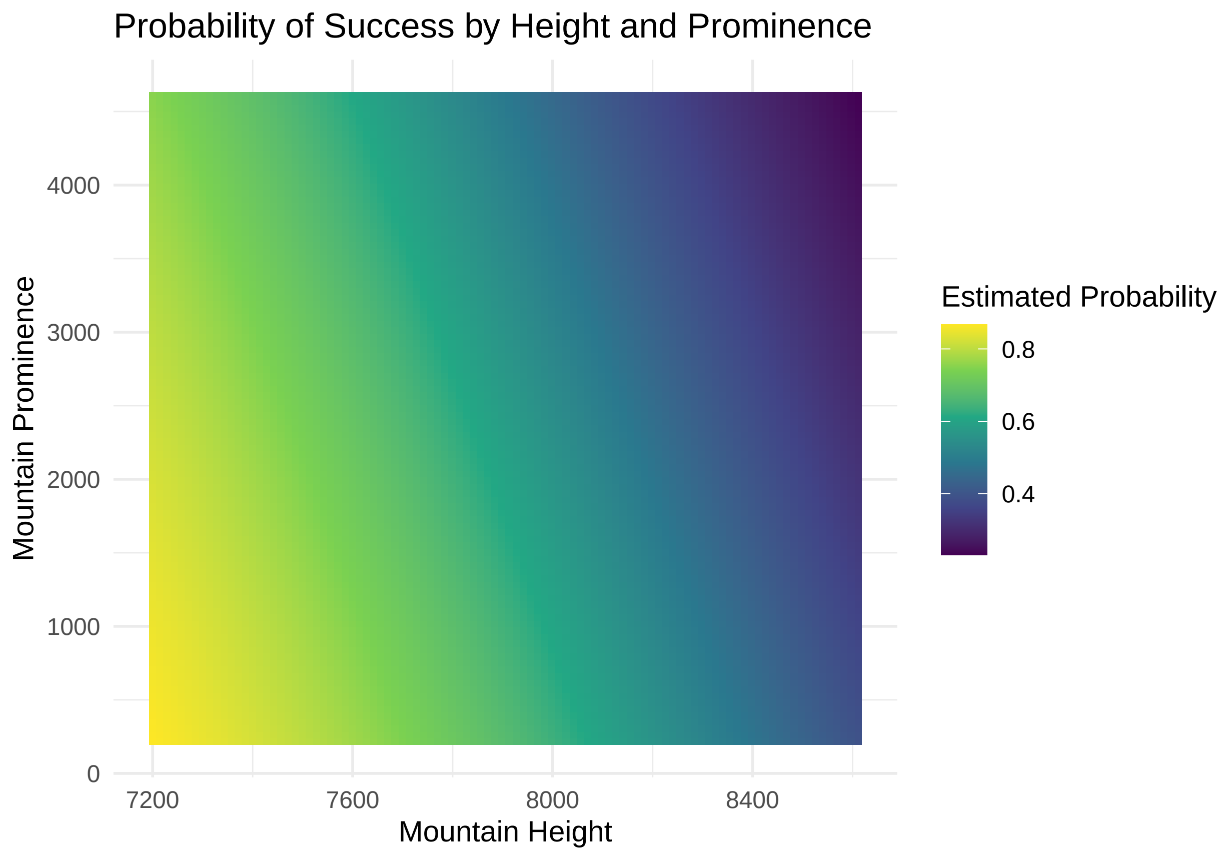
```
library(viridis)
```

```
## Warning: package 'viridis' was built under R version 4.3.3
```

```
## Loading required package: viridisLite
```

```
## Warning: package 'viridisLite' was built under R version 4.3.3
```

```r
numPoints <- 100
heightVal <- seq(from = min(mount$height), to = max(mount$height), length.out = numPoints)
prominenceVal <- seq(from = min(mount$prominence), to = max(mount$prominence), length.out = numPoints)

grid <- expand.grid(height = heightVal, prominence = prominenceVal)

grid$estimatedProb <- numeric(nrow(grid))

for (i in seq_len(nrow(grid))) {
  nu_temp <- as.numeric(
    logitmodel$coefficients[1] +
    logitmodel$coefficients[2] * grid$height[i] +
    logitmodel$coefficients[3] * grid$prominence[i]
  )
  grid$estimatedProb[i] <- exp(nu_temp) / (1 + exp(nu_temp))
}

ggplot(grid, aes(x = height, y = prominence, fill = estimatedProb)) +
  geom_raster() +
  scale_fill_viridis_c(option = "D", name = "Estimated Probability") +  # Use viridis color scale
  labs(
    title = "Probability of Success by Height and Prominence",
    x = "Mountain Height",
    y = "Mountain Prominence"
  ) +
  theme_minimal()
```



The analysis reveals that the estimated probability of success approaches 1 for mountains that are low and have low prominence, whereas it decreases for higher mountains with greater prominence.

(d)

For Mount Everest, we denote the height and prominence as 8848. To compute the probability of a successful ascent, we utilize the linear predictor derived from the coefficients of our logistic regression model. This predictor can be expressed as:

$$\hat{\eta}_e = \hat{\beta} x_e^T = \hat{\beta}_0 + x_{e,\text{height}} \hat{\beta}_{\text{height}} + x_{e,\text{prominence}} \hat{\beta}_{\text{prominence}}$$

Given that the coefficients exhibit asymptotic normality, we have:

$$\hat{\eta}_e \sim N(\mu_{\eta_e} = \beta x_e^T, \sigma_{\eta_e}^2 = x_e \text{Cov}(\beta) x_e^T).$$

Now the confidence interval:

$$[\hat{\eta}_e - z_{0.025}\sigma_{\eta_e}, \hat{\eta}_e + z_{0.025}\sigma_{\eta_e}],$$

We apply the transformation associated with probability:

$$\left[\frac{e^{\hat{\eta}_e - z_{0.025}\sigma_{\eta_e}}}{1 + e^{\hat{\eta}_e - z_{0.025}\sigma_{\eta_e}}}, \frac{e^{\hat{\eta}_e + z_{0.025}\sigma_{\eta_e}}}{1 + e^{\hat{\eta}_e + z_{0.025}\sigma_{\eta_e}}} \right]$$

```r
everestHeight <- 8848
everestProminence <- 8848
linPredEverest <- sum(logitmodel$coefficients * c(1, everestHeight, everestProminence))
successProbEverest <- exp(linPredEverest) / (1 + exp(linPredEverest))
varlinPredEverest <- sqrt(t(c(1, everestHeight, everestProminence)) %*% vcov(logitmodel) %*% c(1, evere
ciLinPredEverest <- c(linPredEverest - qnorm(0.975) * varlinPredEverest,
                                 linPredEverest + qnorm(0.975) * varlinPredEverest)
ciProbEverest <- exp(ciLinPredEverest) / (1 + exp(ciLinPredEverest))

highestMountain <- max(mount$height)
avgHeight <- mean(mount$height)
highestProminence <- max(mount$prominence)

successProbEverest
```

```
## [1] 0.08917783
```

```r
ciProbEverest
```

```
## [1] 0.05486572 0.14173033
```

```r
highestMountain
```

```
## [1] 8611
```

```r
avgHeight
```

```
## [1] 7568.283
```

```r
highestProminence
```

```
## [1] 4608
```

Now we do the same for Chogolisa, but we use $x_{\text{chogolisa, height}}$ and $x_{\text{chogolisa, prominence}}$ equal to 7665 and 1624 respectively.

```r
chogolisaHeight <- 7665
chogolisaProminence <- 1624
linPredChogolisa <- sum(logitmodel$coefficients * c(1, chogolisaHeight, chogolisaProminence))
probChogolisa <- exp(linPredChogolisa) / (1 + exp(linPredChogolisa))
linPredVarChogo <- sqrt(t(c(1, chogolisaHeight, chogolisaProminence)) %*% vcov(logitmodel) %*% c(1, cho
ciLinPredChogolisa <- c(
  linPredChogolisa - qnorm(0.975) * linPredVarChogo,
  linPredChogolisa + qnorm(0.975) * linPredVarChogo
)
ciProbChogo <- exp(ciLinPredChogolisa) / (1 + exp(ciLinPredChogolisa))
attempts <- 22
successChogo <- probChogolisa * attempts
ciSuccessChogo <- ciProbChogo * attempts
```

```r
result <- list(
  Probability = probChogolisa,
  ConfidenceIntervalProbability = ciProbChogo,
  Success = successChogo,
  ConfidenceIntervalSuccess = ciSuccessChogo
)
print(result)
```

```
## $Probability
## [1] 0.7042924
##
## $ConfidenceIntervalProbability
## [1] 0.6832255 0.7245232
##
## $Success
## [1] 15.49443
##
## $ConfidenceIntervalSuccess
## [1] 15.03096 15.93951
```

The predicted count of successful climbers is a bit less than the actual number. This fits with what we noticed earlier— the model underestimates the likelihood of success for mountains with low prominence.

Part2: Poisson regression – Eliteserien 2024

(a)

This exercise focuses on the 2024 Eliteserien, where 16 teams compete in a home-and-away format. As of October 1, 191 matches have been played. The goal is to predict the likelihood of Rosenborg winning the championship based on the current scores.

We use a statistical model where the score of the home team (A) against the away team (B) follows a Poisson distribution. The mean score (λ) depends on each team's strength, represented by a single parameter (β_A) for team A and (β_B) for team B), along with an intercept (β_0) and a home advantage parameter (β_{home}). This model defines team strengths and the uncertainty in rankings, specifically focusing on Rosenborg's championship possibilities.

In (a) we test whether the goals scored by the home and away teams are independent, so we create a contingency table and conduct a Pearson's chi-squared (χ^2) test. When setting up this table in R, we'll separate the data by the number of goals scored. It's important to make sure that at least 80% of the cells have a count of 5 or more, and that no cells are empty (have a count of 0), so we cap the maximum amount of goals scored at 3. This will help us properly examine the relationship between the goals scored by both teams.

In our contingency table, the columns will show how many goals the home team scored, while the rows display the goals scored by the away team. For example, the cell at (1,1) indicates the number of matches where both teams scored one goal which was 21 games. (3,3) shows the number of games where both teams scored three goals or more, which is 5. See the table below.

```r
eliteserie$cappedya <- ifelse(!is.na(eliteserie$ya) & eliteserie$ya > 3, 3, eliteserie$ya)
eliteserie$cappedyh <- ifelse(!is.na(eliteserie$yh) & eliteserie$yh > 3, 3, eliteserie$yh)
goals <- table(eliteserie$cappedyh, eliteserie$cappedya)
goals
```

```
##
##       0  1  2  3
##    0 10 17  8  7
##    1 21 21 12 13
##    2  9 15 15  6
##    3 15 11  6  5
```

To analyze this data, we define H_i as the event where the home team scores i goals and A_j as the event where the away team scores j goals. We denote p_i as the probability of the home team scoring i goals and q_j as the probability of the away team scoring j goals. If we assume these events are independent, then p_{ij} (the probability of both events happening) can be calculated as

$$p_{ij} = p_i \cdot q_j.$$

For each combination of scores, we compare the expected number of matches, calculated as

$$n \cdot p_i \cdot q_i,$$

to the actual observed count in the table, noted as k_{ij}. This comparison is central to the Pearson chi-squared test. The test statistic is calculated by summing

$$\frac{k_{ij} - n \cdot p_i \cdot q_i}{n \cdot p_{ij}}$$

for all cells in the table. If the value of the test is bigger than of the chi-statistic then we reject the null hypothesis. From the code below, we notice that our test has a smaller value than the Chi-statistic and we have a p-value of around 0.4 so we keep our null hypothesis and say that the assumption of independence between the goals made by the home and away teams is reasonable.

```
## Chi-squared Statistic:  9.113215

## Degrees of Freedom:  9

## Chi Value:  14.68366

## P-Value:  0.4268898
```

(b)

```
leagueTable <- data.frame(
  team = unique(eliteserie$home),
  points = rep(0, length(unique(eliteserie$home))),
  goalDiff = rep(0, length(unique(eliteserie$home)))
)

for (teamIndex in seq_along(leagueTable$team)) {
  currentTeam <- leagueTable$team[teamIndex]

  homeResults <- eliteserie$yh[eliteserie$home == currentTeam]
  awayResults <- eliteserie$ya[eliteserie$away == currentTeam]

  homeW <- sum(homeResults > eliteserie$ya[eliteserie$home == currentTeam], na.rm = TRUE)
  awayW <- sum(awayResults > eliteserie$yh[eliteserie$away == currentTeam], na.rm = TRUE)

  homeDraws <- sum(homeResults == eliteserie$ya[eliteserie$home == currentTeam], na.rm = TRUE)
  awayDraws <- sum(awayResults == eliteserie$yh[eliteserie$away == currentTeam], na.rm = TRUE)

  leagueTable$points[teamIndex] <- 3 * (homeW + awayW) + (homeDraws + awayDraws)

  forHome <- sum(homeResults, na.rm = TRUE)
  forAway <- sum(awayResults, na.rm = TRUE)

  againstHome <- sum(eliteserie$ya[eliteserie$home == currentTeam], na.rm = TRUE)
  againstAway <- sum(eliteserie$yh[eliteserie$away == currentTeam], na.rm = TRUE)

  totalScored <- forHome + forAway
  totalConceded <- againstHome + againstAway

  leagueTable$goalDiff[teamIndex] <- totalScored - totalConceded
}
leagueTable <- leagueTable[order(-leagueTable$points, -leagueTable$goalDiff), ]
row.names(leagueTable) <- seq_len(nrow(leagueTable))
leagueTable
```

```
##                   team points goalDiff
## 1          Bodø/Glimt     53       36
## 2               Brann     46       14
## 3               Molde     44       26
## 4              Viking     43       15
## 5           Rosenborg     40        6
## 6         Fredrikstad     40        3
## 7                KFUM     33        1
## 8              HamKam     29        1
## 9              Tromsø     28       -5
## 10        Strømsgodset     28       -9
## 11        Kristiansund     26       -9
## 12        Sarpsborg 08     26      -15
## 13           Haugesund     23      -14
## 14 Sandefjord Fotball     22       -8
## 15                 Odd     22      -18
## 16          Lillestrøm     21      -24
```

A team gets 3 points for a win, 1 point for a draw and 0 points for a loss. Above we see how the table looks as of October 1st.

(c)

We perform the Poisson regression with the intercept, home advantage and each teams' strength as the parameters. It will be written in this form:

$$\ln E(Y) = \beta_0 + \beta_{\text{home}} x_{\text{home}} + \beta_{Bod\varnothing Glimt} x_{\text{BodøGlimt}} + \ldots + \beta_{Vålerenga} x_{\text{Vålerenga}}.$$

In this context, x_{home} is defined as 1 if the score Y corresponds to the home team, and 0 if it corresponds to the away team. For example, if team A is the home team and team B is the away team, then:

- If the score $Y $is for team A:
  - $x_A = 1$
  - $x_B = -1$
- The covariates for all other teams are set to 0.

```r
validVals <- !is.na(eliteserie$yh) & !is.na(eliteserie$ya)
gameNumber <- sum(validVals)

X <- matrix(0, nrow = 2 * gameNumber, ncol = 18)
X[, 1] <- rep(1, 2 * gameNumber)
colnames(X) <- c("Intercept", "HomeAdvantage", as.character(leagueTable$team))
gameIndex <- 1
for (i in which(validVals)) {
  X[gameIndex, 2] <- 1
  X[gameIndex, colnames(X) == eliteserie$home[i]] <- 1
  X[gameIndex, colnames(X) == eliteserie$away[i]] <- -1
  gameIndex <- gameIndex + 1
}
for (i in which(validVals)) {
  X[gameIndex, 2] <- 0
  X[gameIndex, colnames(X) == eliteserie$home[i]] <- -1
  X[gameIndex, colnames(X) == eliteserie$away[i]] <- 1
  gameIndex <- gameIndex + 1
}

Y <- c(eliteserie$yh[validVals], eliteserie$ya[validVals])
```

Now we define the log-likelihood function to estimate the parameters of our model. The likelihood function $L(\beta)$ is :

$$L(\beta) = \prod_{i=1}^{n} \frac{\lambda_i^{y_i}}{y_i!} e^{-\lambda_i},$$

where y_i represents the observed counts.

From this, we find the log-likelihood $l(\beta)$:

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

Here, λ_i is defined in terms of the linear predictor η_i:

$$\lambda_i = e^{\eta_i} = e^{x_i^T \beta},$$

where x_i is the vector of covariates for the i-th observation and β is the vector of parameters we wish to estimate. So now we can code using the optim() function.

```r
logLike <- function(BetaOpt, response, design, strength) {
    BetaCombined <- c(BetaOpt, strength)
    logSum <- 0

    for (index in seq_along(response)) {
        linComb <- exp(design[index, ] %*% BetaCombined)
        logSum <- logSum + response[index] * log(linComb) - linComb - lfactorial(response[index])
    }
    return(-logSum)
}
```

```r
initial <- seq_len(ncol(X) - 1)
optRes <- optim(
  par = initial,
  fn = logLike,
  design = X,
  response = Y,
  strength = 1,
  method = "BFGS"
)
beta <- c(optRes$par, 1)
names(beta) <- colnames(X)
```

```r
beta
```

```
##          Intercept      HomeAdvantage         Bodø/Glimt              Brann
##          0.2311579          0.1553377          1.8166620          1.5244605
##              Molde             Viking          Rosenborg        Fredrikstad
##          1.6812989          1.5648760          1.4060873          1.3806268
##               KFUM             HamKam             Tromsø        Strømsgodset
##          1.3751691          1.3620964          1.2574728          1.1978633
##       Kristiansund       Sarpsborg 08          Haugesund Sandefjord Fotball
##          1.2008263          1.1036945          1.1426724          1.2167559
##                Odd         Lillestrøm
##          1.0707991          1.0000000
```

```r
sortBeta <- sort(beta[-c(1, 2)], decreasing = TRUE)
results <- data.frame(Beta = sortBeta)
print(results)
```

```
##                         Beta
## Bodø/Glimt         1.816662
## Molde              1.681299
## Viking             1.564876
## Brann              1.524460
## Rosenborg          1.406087
## Fredrikstad        1.380627
## KFUM               1.375169
## HamKam             1.362096
## Tromsø             1.257473
## Sandefjord Fotball 1.216756
## Kristiansund       1.200826
## Strømsgodset       1.197863
## Haugesund          1.142672
## Sarpsborg 08       1.103695
## Odd                1.070799
## Lillestrøm         1.000000
```

Our estimated rankings and parameters above indicate some differences from the league table shown in part (b). This variation can be attributed to the last matches of the season, as teams scheduled to play against weaker opponents are likely to be predicted to perform better than those facing tougher competition.

(d)

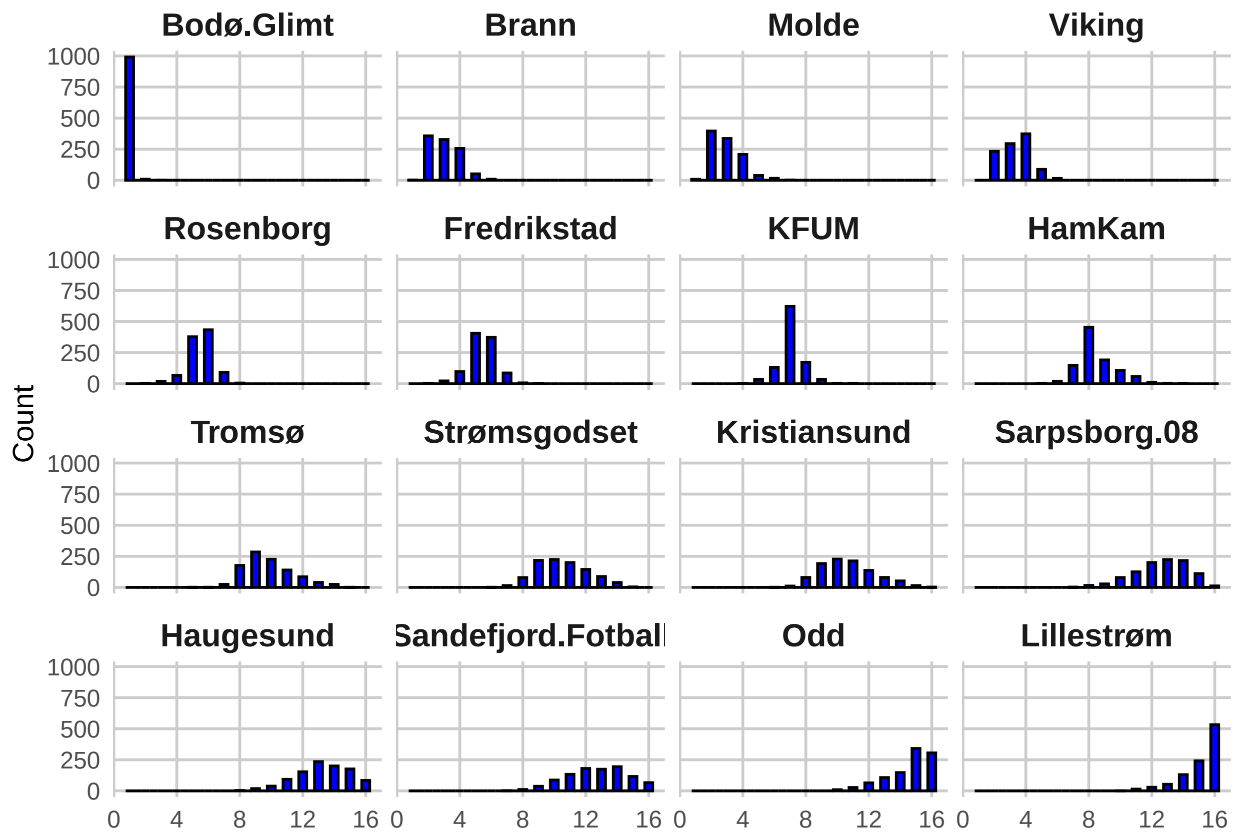
Next, we generate a new design matrix to simulate upcoming results and calculate the value of lambda. The calculation for lambda is shown below. After obtaining this value, we will conduct 1000 simulations of the match scores based on a Poisson distribution.

$$\tilde{\lambda}_{\text{new}} = X\text{new} \cdot \tilde{\beta}$$

```r
Unplayed <- eliteserie[is.na(eliteserie$yh), c("home", "away")]

numMatches <- 2 * nrow(Unplayed)
Xd <- matrix(0, nrow = numMatches, ncol = 18)
Xd[, 1] <- rep(1, numMatches)
colnames(Xd) <- c("intercept", "homeAdvantage", as.character(leagueTable$team))
for (matchIndex in seq_len(nrow(Unplayed))) {
  Xd[matchIndex, 2] <- 1
  Xd[matchIndex, colnames(Xd) == Unplayed$home[matchIndex]] <- 1
  Xd[matchIndex, colnames(Xd) == Unplayed$away[matchIndex]] <- -1
}
for (matchIndex in seq_len(nrow(Unplayed))) {
  Xd[matchIndex + nrow(Unplayed), 2] <- 0
  Xd[matchIndex + nrow(Unplayed), colnames(Xd) == Unplayed$home[matchIndex]] <- -1
  Xd[matchIndex + nrow(Unplayed), colnames(Xd) == Unplayed$away[matchIndex]] <- 1
}

lambda <- exp(Xd %*% beta)
nMatches <- length(lambda)
N <- 1000

pos <- matrix(0, nrow = N, ncol = 16)
colnames(pos) <- leagueTable$team
rownames(pos) <- seq_len(N)
for (simIndex in seq_len(N)) {
  leagueSim <- leagueTable
  Unplayed$yh <- rpois(nMatches / 2, lambda = head(lambda, n = (nMatches / 2)))
  Unplayed$ya <- rpois(nMatches / 2, lambda = tail(lambda, n = (nMatches / 2)))

  for (teamIndex in seq_len(nrow(leagueSim))) {
    leagueSim$points[teamIndex] <- leagueSim$points[teamIndex] +
      sum(3 * (Unplayed$yh[Unplayed$home == leagueSim$team[teamIndex]] > Unplayed$ya[Unplayed$home == le
      sum(3 * (Unplayed$ya[Unplayed$away == leagueSim$team[teamIndex]] > Unplayed$yh[Unplayed$away == le
      sum(1 * (Unplayed$ya[Unplayed$away == leagueSim$team[teamIndex]] == Unplayed$yh[Unplayed$away == l
      sum(1 * (Unplayed$yh[Unplayed$home == leagueSim$team[teamIndex]] == Unplayed$ya[Unplayed$home == l

    leagueSim$goalDiff[teamIndex] <- leagueSim$goalDiff[teamIndex] +
      sum(Unplayed$yh[Unplayed$home == leagueSim$team[teamIndex]]) + sum(Unplayed$ya[Unplayed$away == le
      sum(Unplayed$ya[Unplayed$home == leagueSim$team[teamIndex]]) - sum(Unplayed$yh[Unplayed$away == le
  }

  leagueSim <- leagueSim[order(-leagueSim$points, -leagueSim$goalDiff),]
  rownames(leagueSim) <- seq_len(length(leagueTable$team))

  for (posIndex in seq_len(length(leagueSim$team))) {
    pos[simIndex, leagueSim$team[posIndex] == colnames(pos)] <- posIndex
  }
}

avgPos <- colMeans(pos)
avgPos <- t(t(avgPos))
colnames(avgPos) <- "meanPosition"
avgPos <- data.frame(avgPos)
avgPos$oldPosition <- seq_len(16)
lowestPos <- numeric(16)
hightestPos <- numeric(16)
for (index in seq_len(16)) {
  lowestPos[index] <- max(pos[, index])
  hightestPos[index] <- min(pos[, index])
}
avgPos$hightestPos <- hightestPos
avgPos$lowestPos <- lowestPos
avgPos <- avgPos[order(avgPos$meanPosition), , drop = FALSE]
avgPos$team <- rownames(avgPos)
avgPos <- avgPos[, c(5, 1, 2, 3, 4)]
rownames(avgPos) <- seq_len(16)
avgPos
```

```
##                  team meanPosition oldPosition hightestPos lowestPos
## 1          Bodø/Glimt        1.010           1           1         3
## 2               Molde        2.926           3           1         7
## 3               Brann        3.020           2           1         6
## 4              Viking        3.357           4           2         6
## 5         Fredrikstad        5.418           6           2         9
## 6           Rosenborg        5.520           5           2         8
## 7                KFUM        7.064           7           4        11
## 8              HamKam        8.454           8           5        14
## 9              Tromsø        9.811           9           5        15
## 10        Strømsgodset       10.497          10           6        15
## 11        Kristiansund       10.620          11           6        16
## 12        Sarpsborg 08       12.590          12           7        16
## 13 Sandefjord Fotball       12.702          14           7        16
## 14           Haugesund       13.264          13           8        16
## 15                 Odd       14.599          15          10        16
## 16          Lillestrøm       15.148          16          10        16
```

```
data <- melt(data.frame(pos))
histogram <- ggplot(data = data) +
  geom_histogram(aes(x = value), binwidth = 0.5, fill = "blue", color = "black") +
  facet_wrap(~ variable) +
  labs(x = NULL, y = "Count") +
  theme_minimal() +
  theme(
    strip.background = element_blank(),
    strip.text = element_text(size = 12, face = "bold"),
    panel.grid.major = element_line(color = "grey80"),
    panel.grid.minor = element_blank()
  )
print(histogram)
```



The outcomes of the simulations indicate that Rosenborg is expected to finish either fifth or sixth in this year's Eliteserien. Bodø appears to be the strong favorite to secure the championship, while Lillestrøm is likely facing relegation. Most teams are predicted to remain in similar positions, with a few notable exceptions.

For instance, Sandefjord, currently ranked 14th, is anticipated to rise to 12th place due to a relatively easier schedule compared to their nearby competitors. The teams at both the top and bottom of the standings are likely to maintain their current positions, while those in the middle remain unpredictable. This unpredictability can be attributed to the similar β parameters among these mid-table teams.