Problem 1: Predict the Probabilities of Heart Disease for Different Levels of Snoring Using a Logistic Model

We will analyze the relationship between snoring levels and the incidence of heart disease using a logistic regression model.

Step 1: Create the Data

We create a data frame containing the number of heart disease cases and the total number of subjects at different levels of snoring. This data will serve as the basis for modeling the probability of heart disease across varying snoring levels.

```r
# Define the data
snoring <- c(0, 2, 4, 5)  # Snoring levels
cases <- c(24, 35, 21, 30)  # Number of heart disease cases
subjects <- c(804, 223, 135, 36)  # Total number of subjects

# Create a data frame
data <- data.frame(snoring, cases, subjects)

# View the data
print(data)
```

```
##   snoring cases subjects
## 1       0    24      804
## 2       2    35      223
## 3       4    21      135
## 4       5    30       36
```

Step 2: Fit the Logistic Regression Model

```r
# Fit the logistic regression model
model <- glm(cbind(cases, subjects - cases) ~ snoring, data = data, family = binomial)

# Display the summary of the model
summary(model)
```

```
##
## Call:
## glm(formula = cbind(cases, subjects - cases) ~ snoring, family = binomial,
##     data = data)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.40098    0.17965  -18.93   <2e-16 ***
## snoring      0.65628    0.05941   11.05   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 176.126  on 3  degrees of freedom
## Residual deviance:  43.194  on 2  degrees of freedom
## AIC: 65.611
##
## Number of Fisher Scoring iterations: 4
```

Step 3: Predict Probabilities of Heart Disease for Different Levels of Snoring

```r
# Predict probabilities for the observed snoring levels
data$predicted_prob <- predict(model, type = "response")

# View the updated data
print(data)
```

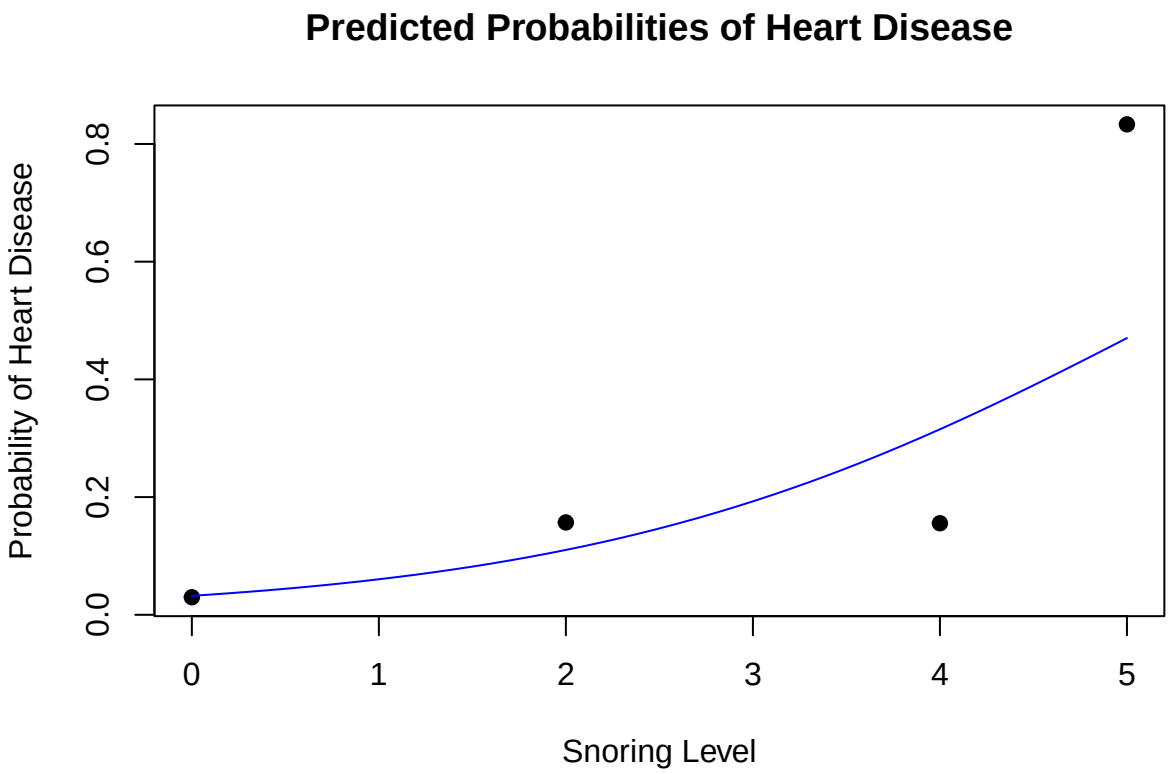
```
##   snoring cases subjects predicted_prob
## 1       0    24      804     0.03226484
## 2       2    35      223     0.11022700
## 3       4    21      135     0.31521098
## 4       5    30       36     0.47013792
```

We can also predict probabilities for a range of snoring levels

```r
# Create a sequence of snoring levels
snoring_levels <- seq(0, 5, by = 0.1)

# Predict probabilities for these snoring levels
predicted_probabilities <- predict(model, newdata = data.frame(snoring = snoring_levels), type = "respo

# Create a data frame of the predicted probabilities
prediction_data <- data.frame(snoring = snoring_levels, probability = predicted_probabilities)
```

Step 4: Plot the Predicted Probabilities

```r
# Plot the observed probabilities
plot(data$snoring, data$cases / data$subjects,
     pch = 19, xlab = "Snoring Level", ylab = "Probability of Heart Disease",
     main = "Predicted Probabilities of Heart Disease")

# Add the predicted probabilities line
lines(prediction_data$snoring, prediction_data$probability, col = "blue")
```



Problem 2: Logistic Regression Without Using `glm()` Function

We will manually write the likelihood function and optimize it using the `optim()` function.

Step 1: Define the Negative Log-Likelihood Function

```r
# Negative log-likelihood function
neg_log_likelihood <- function(params, data) {
  alpha <- params[1]
  beta <- params[2]
  snoring <- data$snoring
  cases <- data$cases
  subjects <- data$subjects

  # Calculate probabilities
  eta <- alpha + beta * snoring
  p <- exp(eta) / (1 + exp(eta))

  # Calculate negative log-likelihood
  nll <- -sum(cases * log(p) + (subjects - cases) * log(1 - p))
  return(nll)
}
```

Step 3: Predict Probabilities Using the Estimated Parameters

Using the optimized values of `alpha` and `beta` obtained from the `optim()` function, we predict the probabilities of the outcome (e.g., heart disease) for given values of the predictor variable. These predictions are made by applying the logistic function with the estimated parameters.

```
# Initial guesses for alpha and beta
initial_params <- c(alpha = 0, beta = 0)

# Optimize the negative log-likelihood
result <- optim(par = initial_params, fn = neg_log_likelihood, data = data, hessian = TRUE)

# Extract the estimated parameters
alpha_hat <- result$par[1]
beta_hat <- result$par[2]

# Display the estimated parameters
cat("Estimated alpha:", alpha_hat, "\n")
```

```
## Estimated alpha: -3.400764
```

```
cat("Estimated beta:", beta_hat, "\n")
```

```
## Estimated beta: 0.6561027
```

Step 4: Compare with `glm()` Estimates

```
# Calculate predicted probabilities
data$manual_prob <- exp(alpha_hat + beta_hat * data$snoring) / (1 + exp(alpha_hat + beta_hat * data$sno

# View the updated data
print(data)
```

```
##   snoring cases subjects predicted_prob manual_prob
## 1       0    24      804     0.03226484  0.03227159
## 2       2    35      223     0.11022700  0.11021383
## 3       4    21      135     0.31521098  0.31510634
## 4       5    30       36     0.47013792  0.46997349
```

Part 2a: For Alpha = -3.86625, Find the Value of Beta That Maximizes the Likelihood

Given `alpha = -3.86625`, we need to find the `beta` that maximizes the likelihood.

Step 1: Define the Negative Log-Likelihood Function with Fixed Alpha

We define the negative log-likelihood function with `alpha` set to `-3.86625` and only `beta` as the variable parameter. This function will be minimized to find the optimal `beta` value that maximizes the likelihood.

```r
# Fixed alpha value
alpha_fixed <- -3.86625

# Negative log-likelihood function with fixed alpha
neg_log_likelihood_beta <- function(beta, data, alpha) {
  snoring <- data$snoring
  cases <- data$cases
  subjects <- data$subjects

  # Calculate probabilities
  eta <- alpha + beta * snoring
  p <- exp(eta) / (1 + exp(eta))

  # Calculate negative log-likelihood
  nll <- -sum(cases * log(p) + (subjects - cases) * log(1 - p))
  return(nll)
}
```

Step 2: Compute Negative Log-Likelihood for a Range of Beta Values

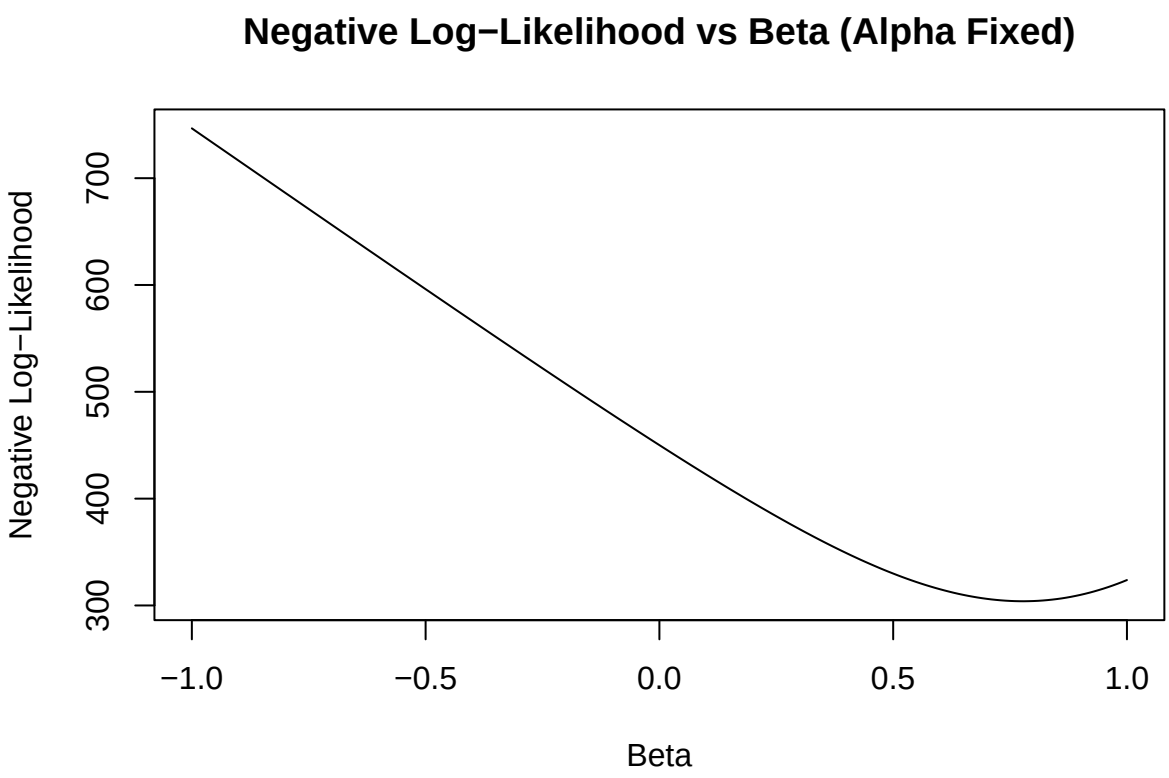
We calculate the negative log-likelihood over a range of `beta` values, keeping `alpha` fixed at `-3.86625`. This helps us observe how the likelihood changes with `beta` and identify the value that maximizes it.

```r
# Define a range of beta values
beta_values <- seq(-1, 1, by = 0.01)

# Calculate negative log-likelihood for each beta value
nll_values <- sapply(beta_values, function(beta) neg_log_likelihood_beta(beta, data, alpha_fixed))
```

Step 3: Plot the Negative Log-Likelihood Function

We will plot the negative log-likelihood function against the range of `beta` values to visualize the point at which the likelihood is maximized. The minimum point on this plot corresponds to the optimal `beta` value.

```r
# Plot the negative log-likelihood vs beta
plot(beta_values, nll_values, type = "l",
     xlab = "Beta", ylab = "Negative Log-Likelihood",
     main = "Negative Log-Likelihood vs Beta (Alpha Fixed)")
```



Step 4: Find the Beta That Minimizes the Negative Log-Likelihood

Using the `optim()` function, we find the value of `beta` that minimizes the negative log-likelihood, with `alpha` fixed at `-3.86625`. This `beta` value maximizes the likelihood for the given `alpha`.

```
# Optimize to find the beta that minimizes the negative log-likelihood
opt_result_beta <- optimize(f = neg_log_likelihood_beta, interval = c(-1, 1), data = data, alpha = alpha

# Estimated beta
beta_hat_fixed_alpha <- opt_result_beta$minimum

# Display the estimated beta
cat("Estimated beta (with alpha fixed at", alpha_fixed, "):", beta_hat_fixed_alpha, "\n")
```

```
## Estimated beta (with alpha fixed at -3.86625 ): 0.777901
```

Part 2b: For Beta = 0.39734, Find the Value of Alpha That Maximizes the Likelihood

Given `beta = 0.39734`, we need to find the `alpha` that maximizes the likelihood.

Step 1: Define the Negative Log-Likelihood Function with Fixed Beta

We define the negative log-likelihood function with `beta` set to `0.39734` and only `alpha` as the variable parameter. This function will be minimized to find the optimal `alpha` value that maximizes the likelihood.

```r
# Fixed beta value
beta_fixed <- 0.39734

# Negative log-likelihood function with fixed beta
neg_log_likelihood_alpha <- function(alpha, data, beta) {
  snoring <- data$snoring
  cases <- data$cases
  subjects <- data$subjects

  # Calculate probabilities
  eta <- alpha + beta * snoring
  p <- exp(eta) / (1 + exp(eta))

  # Calculate negative log-likelihood
  nll <- -sum(cases * log(p) + (subjects - cases) * log(1 - p))
  return(nll)
}
```

Step 2: Compute Negative Log-Likelihood for a Range of Alpha Values

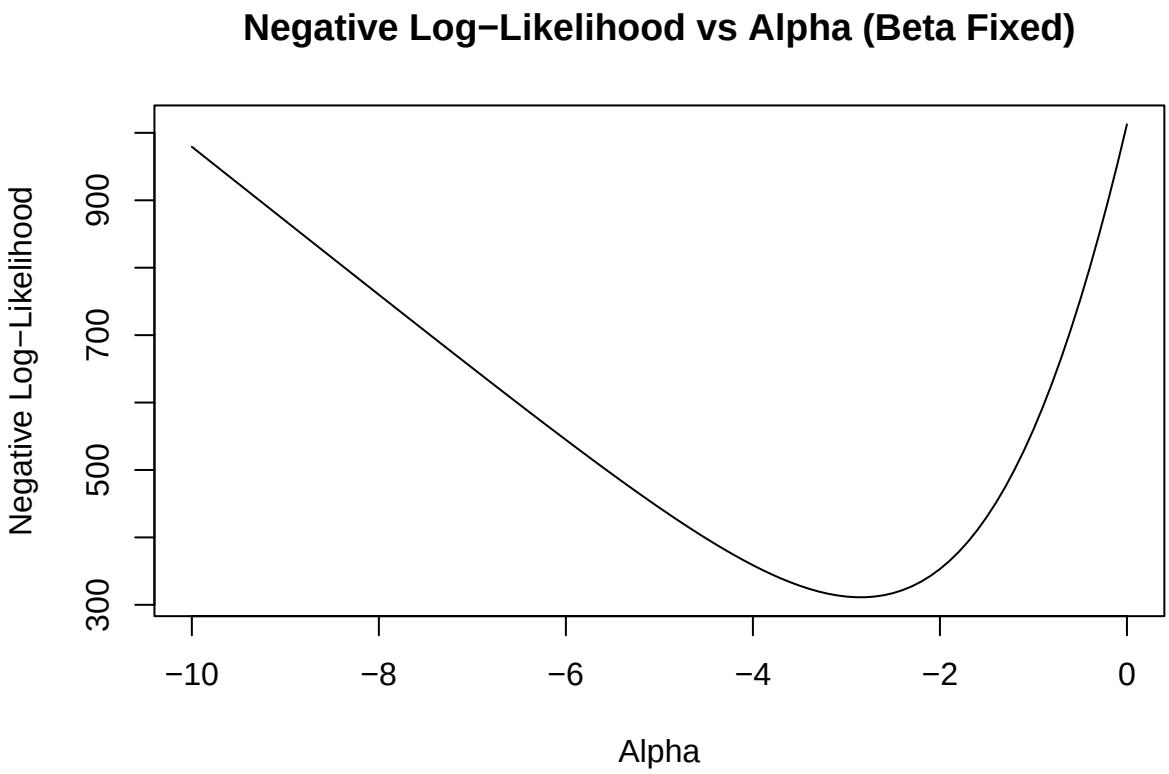
We calculate the negative log-likelihood over a range of `alpha` values, keeping `beta` fixed at `0.39734`. This allows us to observe how the likelihood changes with `alpha` and identify the value that maximizes it.

```r
# Define a range of alpha values
alpha_values <- seq(-10, 0, by = 0.1)

# Calculate negative log-likelihood for each alpha value
nll_values_alpha <- sapply(alpha_values, function(alpha) neg_log_likelihood_alpha(alpha, data, beta_fix
```

Step 3: Plot the Negative Log-Likelihood Function

We will plot the negative log-likelihood function against the range of `alpha` values to visually identify the point where the likelihood is maximized. The minimum point on this plot corresponds to the optimal `alpha` value.

```r
# Plot the negative log-likelihood vs alpha
plot(alpha_values, nll_values_alpha, type = "l",
     xlab = "Alpha", ylab = "Negative Log-Likelihood",
     main = "Negative Log-Likelihood vs Alpha (Beta Fixed)")
```



Step 4: Find the Alpha That Minimizes the Negative Log-Likelihood

Using the `optim()` function, we find the value of `alpha` that minimizes the negative log-likelihood, with `beta` fixed at `0.39734`. This `alpha` value maximizes the likelihood for the given `beta`.

```
# Optimize to find the alpha that minimizes the negative log-likelihood
opt_result_alpha <- optimize(f = neg_log_likelihood_alpha, interval = c(-10, 0), data = data, beta = be

# Estimated alpha
alpha_hat_fixed_beta <- opt_result_alpha$minimum

# Display the estimated alpha
cat("Estimated alpha (with beta fixed at", beta_fixed, "):", alpha_hat_fixed_beta, "\n")
```

```
## Estimated alpha (with beta fixed at 0.39734 ): -2.847634
```

Problem: Comparison of Different Binary Regression Models on Heart Disease Data

We will apply different binary regression models to the dataset used in the previous analysis, which examines the relationship between snoring levels and the incidence of heart disease. Specifically, we will:

1. Apply a **probit model**.
2. Apply a **complementary log-log model**.
3. Apply a **log-log model**.
4. Predict the number of heart disease patients in each class for each model.
5. Compare the models using **Mean Squared Error (MSE)**.
6. Calculate any two **goodness-of-fit statistics**.

Step 1: Data Preparation

We will use the same dataset as before, containing snoring levels, number of heart disease cases, and total subjects in each category.

```
# Define the data
snoring <- c(0, 2, 4, 5)        # Snoring levels
cases <- c(24, 35, 21, 30)      # Number of heart disease cases
subjects <- c(804, 223, 135, 36) # Total number of subjects

# Create a data frame
data <- data.frame(snoring, cases, subjects)

# Calculate observed probabilities
data$observed_prob <- data$cases / data$subjects

# View the data
print(data)
```

```
##   snoring cases subjects observed_prob
## 1       0    24      804    0.02985075
## 2       2    35      223    0.15695067
## 3       4    21      135    0.15555556
## 4       5    30       36    0.83333333
```

Step 2: Apply Different Binary Regression Models

2.1 Probit Model

We will fit a probit regression model using the `glm()` function with `family = binomial(link = "probit")`. This model uses a probit link function to estimate the probability of heart disease based on snoring levels.

```
# Fit the probit model
probit_model <- glm(cbind(cases, subjects - cases) ~ snoring, data = data, family = binomial(link = "pr

# Display the summary of the model
summary(probit_model)
```

```
##
## Call:
## glm(formula = cbind(cases, subjects - cases) ~ snoring, family = binomial(link = "probit"),
##     data = data)
##
## Coefficients:
```

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.86621    0.08112  -23.01   <2e-16 ***
## snoring      0.34650    0.03100   11.18   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 176.126  on 3  degrees of freedom
## Residual deviance:  43.829  on 2  degrees of freedom
## AIC: 66.246
##
## Number of Fisher Scoring iterations: 4
```

2.2 Complementary Log-Log Model

We will fit a complementary log-log (cloglog) model using `family = binomial(link = "cloglog")`. This model applies a complementary log-log link function to estimate the probability of heart disease based on snoring levels.

```r
# Fit the complementary log-log model
cloglog_model <- glm(cbind(cases, subjects - cases) ~ snoring, data = data, family = binomial(link = "cl

# Display the summary of the model
summary(cloglog_model)
```

```
##
## Call:
## glm(formula = cbind(cases, subjects - cases) ~ snoring, family = binomial(link = "cloglog"),
##     data = data)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.41944    0.17439  -19.61   <2e-16 ***
## snoring      0.61509    0.05241   11.74   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 176.126  on 3  degrees of freedom
## Residual deviance:  41.042  on 2  degrees of freedom
## AIC: 63.46
##
## Number of Fisher Scoring iterations: 6
```

2.3 Log-Log Model

Since R does not have a built-in log-log link function in the binomial family, we can define a custom link function to implement the log-log model. This allows us to model the probability of heart disease using a log-log transformation based on snoring levels.

```r
# Define the log-log link function
loglog_link <- function() {
  linkfun <- function(mu) -log(-log(mu))
  linkinv <- function(eta) exp(-exp(-eta))
  mu.eta <- function(eta) exp(-eta - exp(-eta))
  valideta <- function(eta) TRUE
  link <- "loglog"
  structure(list(linkfun = linkfun, linkinv = linkinv, mu.eta = mu.eta,
                 valideta = valideta, name = link), class = "link-glm")
}

# Fit the log-log model
loglog_model <- glm(cbind(cases, subjects - cases) ~ snoring, data = data,
                    family = binomial(link = loglog_link()))

# Display the summary of the model
summary(loglog_model)
```

```
##
## Call:
## glm(formula = cbind(cases, subjects - cases) ~ snoring, family = binomial(link = loglog_link()),
##     data = data)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.25350    0.05449  -23.01   <2e-16 ***
## snoring      0.27364    0.02613   10.47   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 176.126  on 3  degrees of freedom
## Residual deviance:  45.587  on 2  degrees of freedom
## AIC: 68.004
##
## Number of Fisher Scoring iterations: 4
```

Step 3: Predict the Number of Heart Disease Patients for Each Model

We will predict the probabilities and calculate the expected number of heart disease cases for each model.

3.1 Predictions from the Probit Model

Using the fitted probit model, we will predict the probability of heart disease for each snoring level and calculate the expected number of cases.

```r
# Predicted probabilities
data$probit_prob <- predict(probit_model, type = "response")

# Expected number of cases
data$probit_cases <- data$probit_prob * data$subjects
```

3.2 Predictions from the Complementary Log-Log Model

Using the fitted complementary log-log model, we will predict the probability of heart disease for each snoring level and calculate the expected number of cases.

```
# Predicted probabilities
data$cloglog_prob <- predict(cloglog_model, type = "response")

# Expected number of cases
data$cloglog_cases <- data$cloglog_prob * data$subjects
```

3.3 Predictions from the Log-Log Model

Using the fitted log-log model, we will predict the probability of heart disease for each snoring level and calculate the expected number of cases.

```
# Predicted probabilities
data$loglog_prob <- predict(loglog_model, type = "response")

# Expected number of cases
data$loglog_cases <- data$loglog_prob * data$subjects
```

3.4 View the Updated Data

```
# View the data with predictions
print(data)
```

```
##   snoring cases subjects observed_prob probit_prob probit_cases cloglog_prob
## 1       0    24      804    0.02985075  0.03100576     24.92863   0.03220098
## 2       2    35      223    0.15695067  0.12035476     26.83911   0.10595602
## 3       4    21      135    0.15555556  0.31553656     42.59744   0.31835702
## 4       5    30       36    0.83333333  0.44681284     16.08526   0.50783669
##   cloglog_cases loglog_prob loglog_cases
## 1      25.88959   0.0301200     24.21648
## 2      23.62819   0.1318213     29.39615
## 3      42.97820   0.3096658     41.80488
## 4      18.28212   0.4099874     14.75954
```

Step 4: Compare Models Through Mean Squared Error (MSE)

We will calculate the Mean Squared Error (MSE) between the observed and predicted number of heart disease cases for each model. This comparison will help us assess the accuracy of each model in estimating the actual cases.

```
# Calculate MSE for each model
mse_probit <- mean((data$cases - data$probit_cases)^2)
mse_cloglog <- mean((data$cases - data$cloglog_cases)^2)
mse_loglog <- mean((data$cases - data$loglog_cases)^2)

# Display the MSE values
cat("Mean Squared Error (Probit Model):", round(mse_probit, 4), "\n")
```

```
## Mean Squared Error (Probit Model): 181.8829
```

```
cat("Mean Squared Error (Complementary Log-Log Model):", round(mse_cloglog, 4), "\n")
```

```
## Mean Squared Error (Complementary Log-Log Model): 188.3096
```

```
cat("Mean Squared Error (Log-Log Model):", round(mse_loglog, 4), "\n")
```

```
## Mean Squared Error (Log-Log Model): 174.1411
```

Step 5: Calculate Goodness-of-Fit Statistics

We will calculate two goodness-of-fit statistics to evaluate each model's performance: **Deviance** and the **Hosmer-Lemeshow Test**.

5.1 Deviance

The deviance of the model indicates how well the model fits the data compared to a saturated model. Lower deviance values suggest a better fit.

```
# Deviance for each model
deviance_probit <- deviance(probit_model)
deviance_cloglog <- deviance(cloglog_model)
deviance_loglog <- deviance(loglog_model)

# Degrees of freedom
df_probit <- df.residual(probit_model)
df_cloglog <- df.residual(cloglog_model)
df_loglog <- df.residual(loglog_model)

# Display the deviance and degrees of freedom
cat("Deviance (Probit Model):", deviance_probit, "with", df_probit, "degrees of freedom\n")
```

```
## Deviance (Probit Model): 43.82862 with 2 degrees of freedom
```

```
cat("Deviance (Complementary Log-Log Model):", deviance_cloglog, "with", df_cloglog, "degrees of freedo
```

```
## Deviance (Complementary Log-Log Model): 41.04213 with 2 degrees of freedom
```

```
cat("Deviance (Log-Log Model):", deviance_loglog, "with", df_loglog, "degrees of freedom\n")
```

```
## Deviance (Log-Log Model): 45.58676 with 2 degrees of freedom
```

5.2 Hosmer-Lemeshow Test

Since our dataset is small, the Hosmer-Lemeshow test may not be very informative, but we can still perform it for comparison purposes. This test will assess the goodness-of-fit by comparing observed and predicted values across different risk groups. We will use the `ResourceSelection` package to perform the test.

Performing the Hosmer-Lemeshow Test for Each Model:

```r
library(ResourceSelection)
```

```
## ResourceSelection 0.3-6   2023-06-27
```

```r
# For Probit Model
hl_probit <- hoslem.test(data$cases, fitted(probit_model) * data$subjects, g = 4)
cat("Hosmer-Lemeshow Test (Probit Model):\n")
```

```
## Hosmer-Lemeshow Test (Probit Model):
```

```r
print(hl_probit)
```

```
##
##  Hosmer and Lemeshow goodness of fit (GOF) test
##
## data:  data$cases, fitted(probit_model) * data$subjects
## X-squared = -1.1587, df = 2, p-value = 1
```

```r
# For Complementary Log-Log Model
hl_cloglog <- hoslem.test(data$cases, fitted(cloglog_model) * data$subjects, g = 4)
cat("\nHosmer-Lemeshow Test (Complementary Log-Log Model):\n")
```

```
##
## Hosmer-Lemeshow Test (Complementary Log-Log Model):
```

```r
print(hl_cloglog)
```

```
##
##  Hosmer and Lemeshow goodness of fit (GOF) test
##
## data:  data$cases, fitted(cloglog_model) * data$subjects
## X-squared = -0.94973, df = 2, p-value = 1
```

```r
# For Log-Log Model
hl_loglog <- hoslem.test(data$cases, fitted(loglog_model) * data$subjects, g = 4)
cat("\nHosmer-Lemeshow Test (Log-Log Model):\n")
```

```
##
## Hosmer-Lemeshow Test (Log-Log Model):
```

```r
print(hl_loglog)
```

```
##
##  Hosmer and Lemeshow goodness of fit (GOF) test
##
## data:  data$cases, fitted(loglog_model) * data$subjects
## X-squared = -1.4352, df = 2, p-value = 1
```

Note: The Hosmer-Lemeshow test may produce warnings or errors if the data does not have enough variability or if the groupings are not appropriate.

Step 6: Interpretation and Conclusion

6.1 Comparing Mean Squared Errors

```
cat("Mean Squared Error (Probit Model):", round(mse_probit, 4), "\n")
```

```
## Mean Squared Error (Probit Model): 181.8829
```

```
cat("Mean Squared Error (Complementary Log-Log Model):", round(mse_cloglog, 4), "\n")
```

```
## Mean Squared Error (Complementary Log-Log Model): 188.3096
```

```
cat("Mean Squared Error (Log-Log Model):", round(mse_loglog, 4), "\n")
```

```
## Mean Squared Error (Log-Log Model): 174.1411
```

The model with the lowest MSE provides the best fit to the data in terms of predicted number of cases.

6.2 Goodness-of-Fit Statistics

- **Deviance:** Lower deviance values indicate a better fit.
- **Hosmer-Lemeshow Test:** A high p-value (greater than 0.05) suggests that the model fits the data well, indicating good agreement between observed and predicted values across different risk groups.

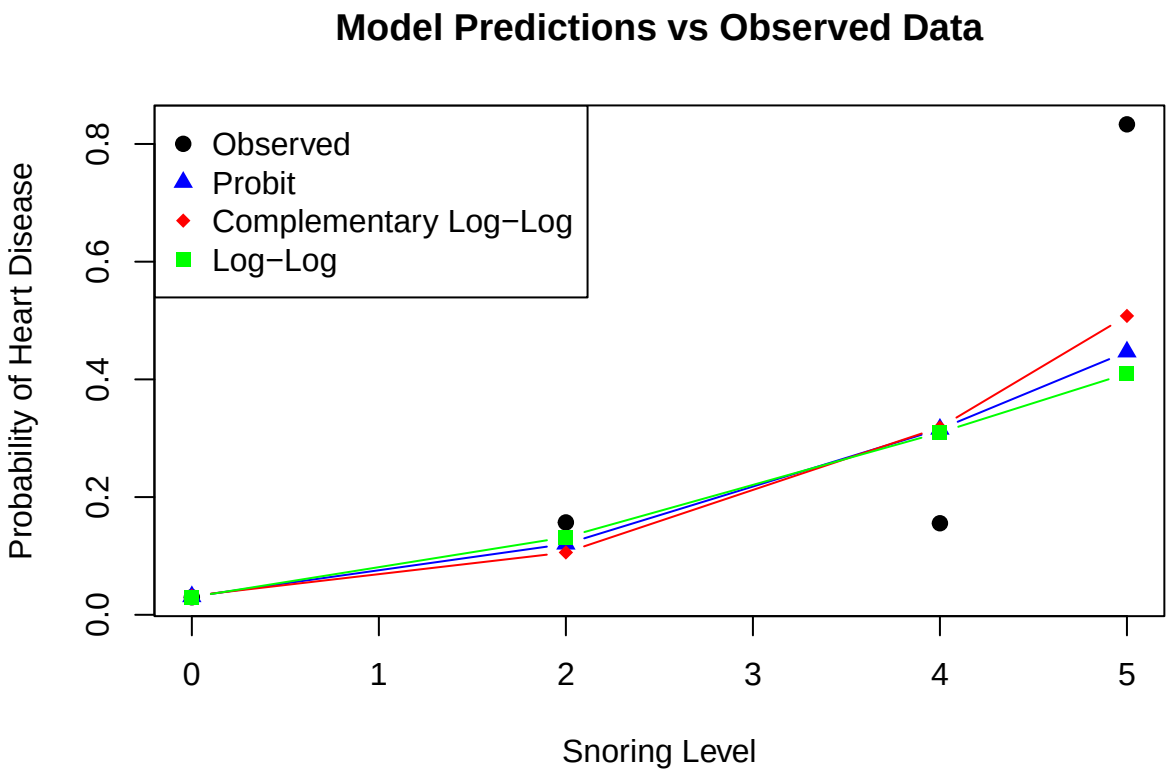
6.3 Model Selection

Based on the MSE and goodness-of-fit statistics, we can determine which model provides the best fit to the data. We should also consider the theoretical justification for choosing a particular link function, as different link functions may offer different interpretations of the relationship between snoring levels and heart disease risk.

6.4 Visual Comparison

We can plot the observed and predicted probabilities for each model to visually assess the fit. This comparison allows us to see how well each model captures the trend in the data across varying snoring levels.

```
# Plot observed probabilities
plot(data$snoring, data$observed_prob, pch = 19, xlab = "Snoring Level", ylab = "Probability of Heart D

# Add predicted probabilities
lines(data$snoring, data$probit_prob, type = "b", col = "blue", pch = 17)
lines(data$snoring, data$cloglog_prob, type = "b", col = "red", pch = 18)
lines(data$snoring, data$loglog_prob, type = "b", col = "green", pch = 15)

# Add legend
legend("topleft", legend = c("Observed", "Probit", "Complementary Log-Log", "Log-Log"), col = c("black"
```



Final Remarks

- **Probit Model:** Assumes a normal distribution of the latent variable, making it useful when the underlying probability distribution is symmetric.
- **Complementary Log-Log Model:** Appropriate for modeling time-to-event data or when the probability of the outcome increases rapidly at first and then slowly approaches 1. This model is often used in survival analysis and other contexts where the risk increases progressively over time.
- **Log-Log Model:** Suitable when the probability of an event decreases rapidly at first and then slowly approaches zero. This model can be useful when the probability of the event behaves in a complementary manner to the complementary log-log model.

By comparing the MSE and goodness-of-fit statistics, along with theoretical considerations and a visual assessment of observed vs. predicted probabilities, we can make an informed decision about the most appropriate model for our data. This comprehensive approach ensures that the selected model not only fits the data well but also aligns with the underlying assumptions and distributional characteristics of the probability of heart disease associated with different snoring levels.