DIY Regression

Jasmine Siswandjo and Saumya Seth

2023-12-17

```
library(knitr)
opts_chunk$set(fig.width = 5, fig.height = 4, fig.align = "left", out.width = "8.5in")

set.seed(123)
library(tidyverse)
library(dplyr)
library(MASS)
library(alr4)
library(pscl)
library(glmbb) # for crabs data
library(kableExtra)
library(lmtest)
```

Linear Regression

Introduction

Linear Regression is one of the simplest regressions out there. In predicting an outcome from various covariate(s), it creates the 'best-fitting' line to the data that we observe to create a model - in that it predicts values on the line when given specific values of the covariates.

Uses

Linear Regression is used across various fields. It is a model which has high bias and low variance. This means that even though it may not fit the data observed in the most optimal way (in that it may not be able to capture complexities in the data), it is not that sensitive to changes in the training data, which can make it more stable when dealing with small fluctuations or noise in the data set. Linear Regression can be used for predicting continuous, categorical, and even binary outcomes (as is often done in Causal Inference).

Assumptions

- The predictors and the outcome are linearly related to one another
- The errors are normally distributed and are independent of one another
- The errors are homoscedastic

Our Linear Regression Implementation

Our Linear Regression implementation: (Note that we use bootstrapping to estimate standard errors)

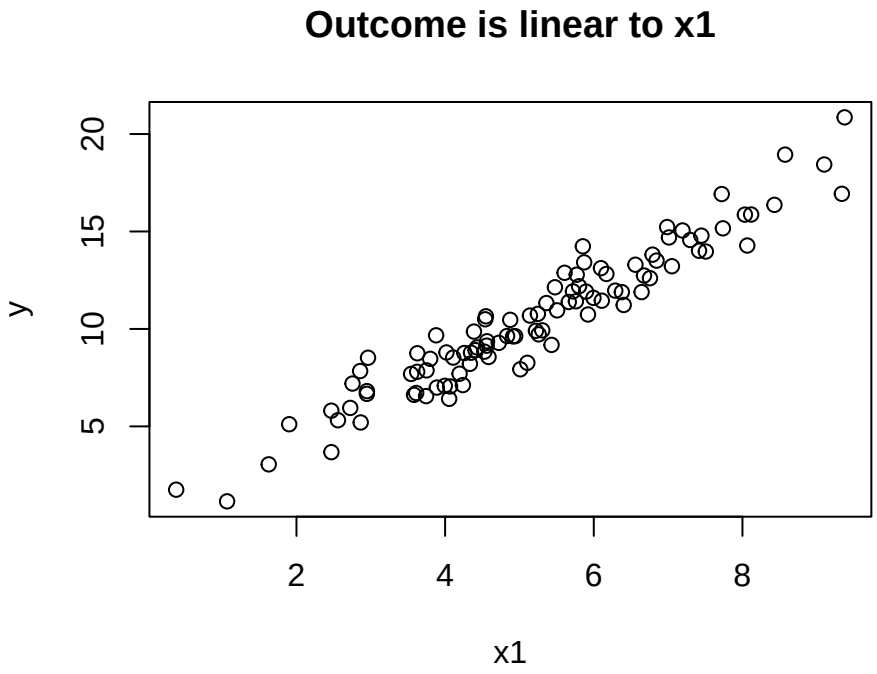
```
linear_regression <- function(data, ..., y) {
  x_parameters <- c(...)
  n <- nrow(data)
  # defining the predictor matrix
```

```r
X <-
  matrix(c(rep(1, n), x_parameters),
    nrow = n,
    ncol = ncol(data)
  )
# defining the outcome matrix
Y <- matrix(y, nrow = n, ncol = 1)
# solving for the beta coefficients
beta <- solve(t(X) %*% X) %*% t(X) %*% Y
# creating a vector 'estimate' for the beta coefficients
estimate <- c()
for (i in 1:ncol(X)) {
  estimate[i] <- beta[i]
}
# bootstrapping to estimate the standard errors
num_bootstraps <- 10000
bootstrap_betas <-
  matrix(0, nrow = num_bootstraps, ncol = ncol(data))
for (i in 1:num_bootstraps) {
  sample_indices <- sample(nrow(data), replace = TRUE)
  bootstrap_data <- data[sample_indices, ]
  bootstrap_X <-
    as.matrix(cbind(1, bootstrap_data[, 1:(ncol(bootstrap_data) - 1)]))
  bootstrap_Y <- as.matrix(bootstrap_data$y, ncol = 1)
  bootstrap_beta <-
    solve(t(bootstrap_X) %*% bootstrap_X) %*% t(bootstrap_X) %*% bootstrap_Y
  bootstrap_betas[i, ] <- bootstrap_beta
}
# finding the standard deviation of the bootstrapped betas to find the
# standard error of the coefficients
se <- c()
for (i in 1:ncol(X)) {
  se[i] <- apply(bootstrap_betas, 2, sd)[i]
}
# calculating the t-statistic
t <- estimate / se
# defining the degrees of freedom
df <- nrow(X) - ncol(X)
# calculating the p-value
p <- 2 * pt(t, df, lower = F)
# calculating the residuals
resid <- Y - X %*% beta
residual <- sqrt(mean((resid)^2))
# defining the row names of the output data frame
rownames <- c()
for (i in 1:((ncol(X)) - 1)) {
  rownames[i] <- i
}
test <- list(
  plot(resid, main = "Residual Plot to test homoscedasticity of errors", ylim = c(-10, 10)),
  qqnorm(resid, main = "Q-Q plot to test normality of errors"),
  pairs(data, main = "Assessing Linearity of\n Predictors with Outcome")
)
```

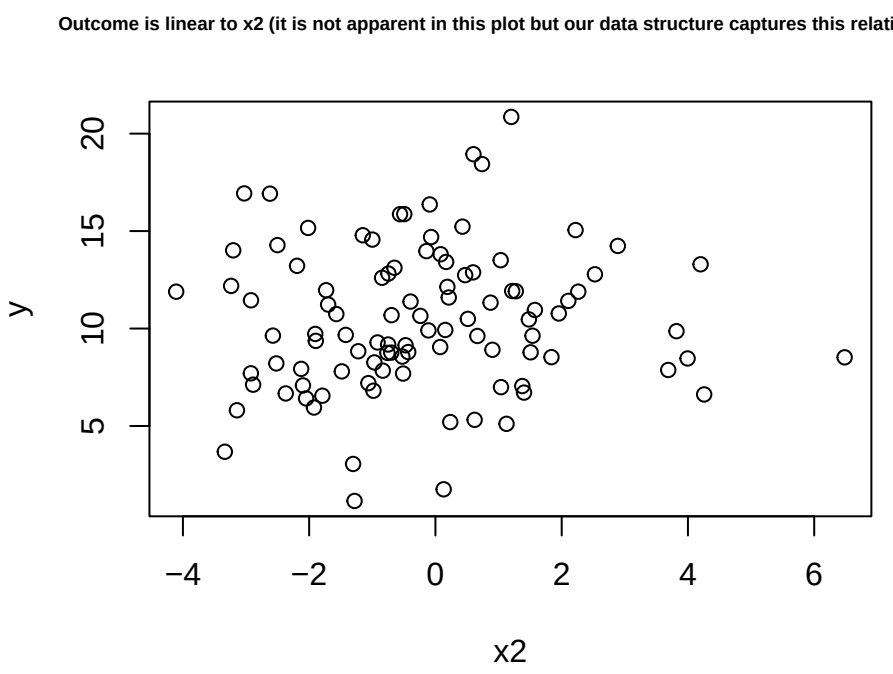
```r
  impl <- data.frame(
    Estimate = estimate,
    Std.Error = se,
    t.value = t,
    p.value = p,
    Residual = c(residual, rep(NA, ncol(X) - 1)),
    DegOfFreedom = c(df, rep(NA, ncol(X) - 1)),
    row.names = c("(Intercept)", paste0(rep("x", ncol(
      X
    ) - 1), rownames))
  )
  # returning a data frame akin to the lm output
  return(list(test, impl))
}
```

Creating a test data set which meets all Linear Regression assumptions to check if our function works.

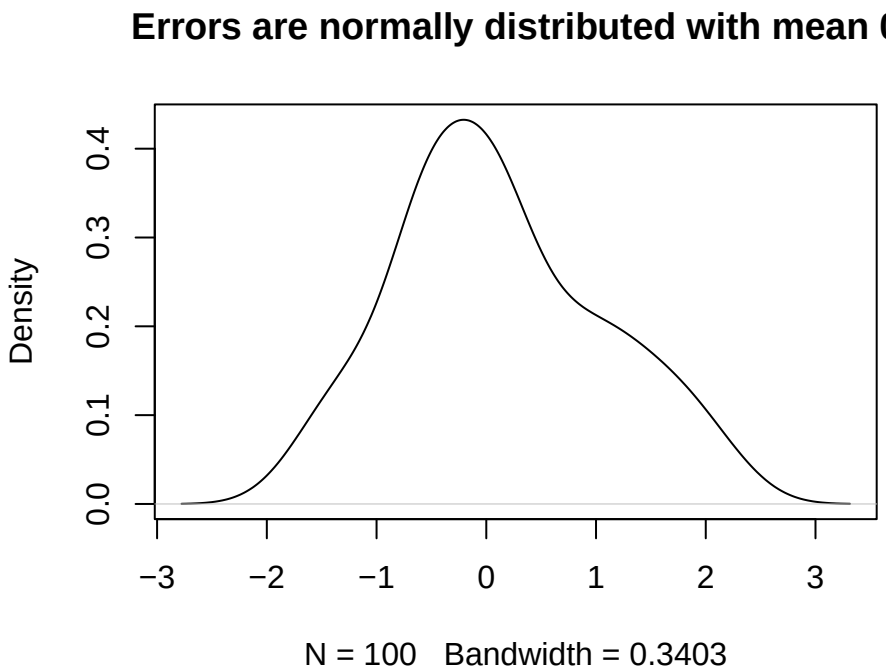
```r
test_linear_regression_data <-
  data.frame(
    x1 = rnorm(100, mean = 5, sd = 2),
    x2 = rnorm(100, mean = 0, sd = 2)
  )
error <- rnorm(100, mean = 0, sd = 1) # errors are homoscedastic
test_linear_regression_data$y <-
  2 * test_linear_regression_data$x1 +
  0.2 * test_linear_regression_data$x2 + error

plot(test_linear_regression_data$x1, test_linear_regression_data$y,
  xlab = "x1", ylab = "y",
  main = "Outcome is linear to x1"
)
```



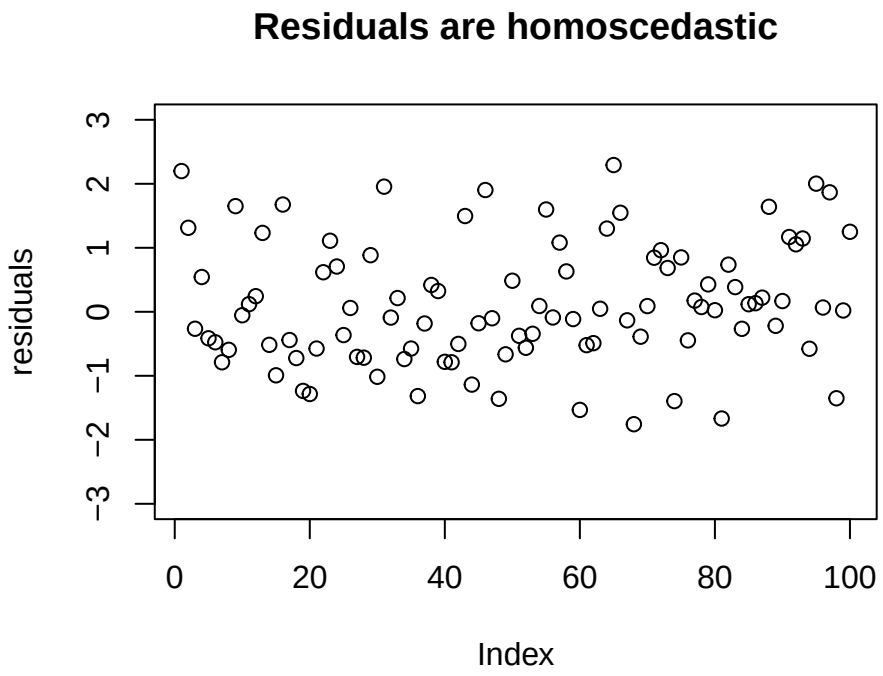
```r
plot(test_linear_regression_data$x2, test_linear_regression_data$y,
  xlab = "x2", ylab = "y",
  main = "Outcome is linear to x2 (it is not apparent in this plot but our data structure captures this
)
```




```r
plot(density(error), main = "Errors are normally distributed with mean 0")
```

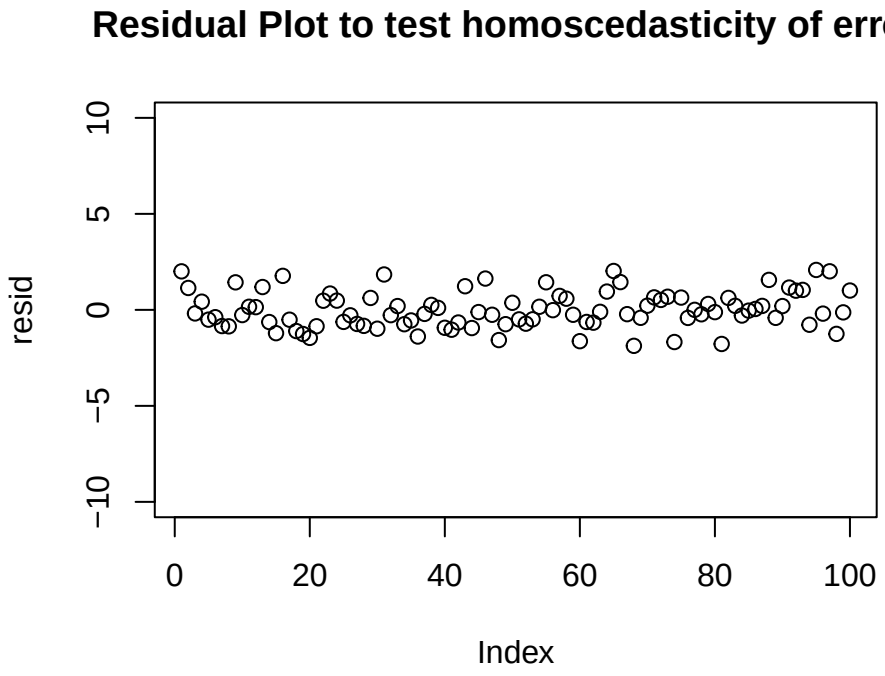


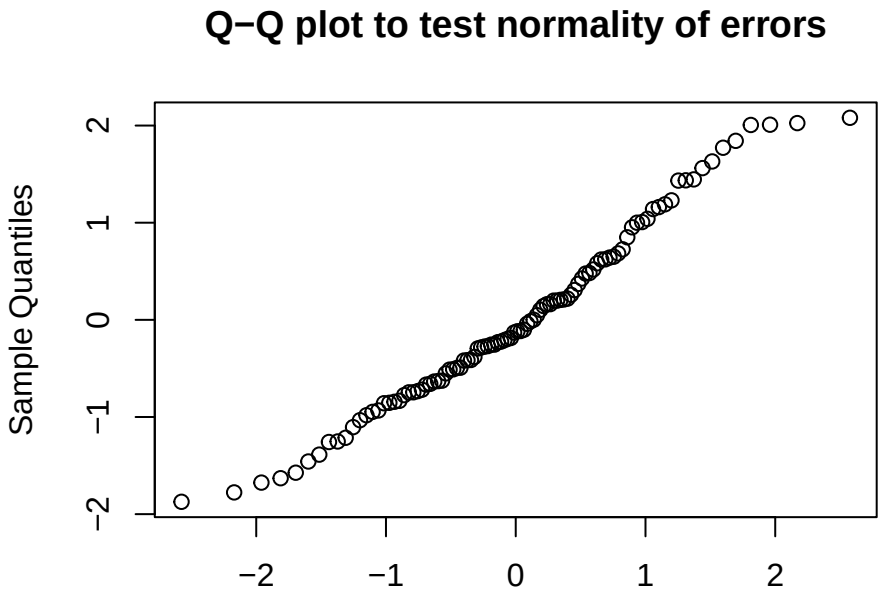

```r
plot(error,
  ylab = "residuals", main = "Residuals are homoscedastic", ylim = c(-3, 3)
)
```

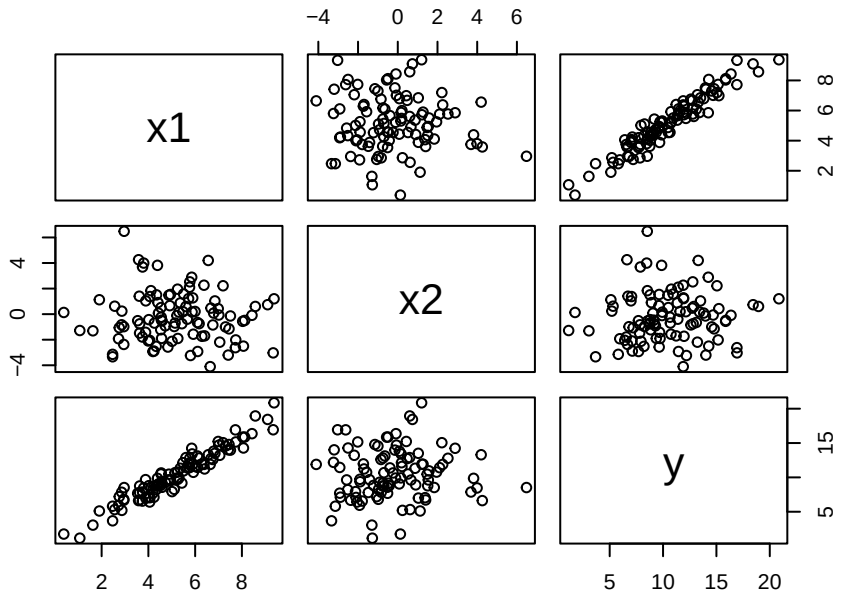


Testing Assumptions for Linear Regression

```
our_implementation <- linear_regression(
  test_linear_regression_data,
  test_linear_regression_data$x1,
  test_linear_regression_data$x2,
  y = test_linear_regression_data$y
)[[2]]
```








```
our_implementation
```

```
##               Estimate  Std.Error   t.value      p.value  Residual DegOfFreedom
## (Intercept) 0.4679943 0.31344739  1.493055 1.386684e-01 0.9369198            97
## x1          1.9334142 0.05792076 33.380331 5.582525e-55        NA            NA
## x2          0.2119056 0.05038448  4.205772 5.802633e-05        NA            NA
```

Comparing our output to R's output.

```
r_implementation <-
  summary(lm(y ~ x1 + x2, data = test_linear_regression_data))
```

```
r_implementation
```

```
##
## Call:
## lm(formula = y ~ x1 + x2, data = test_linear_regression_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.8730 -0.6607 -0.1245  0.6214  2.0798
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.46799    0.28753   1.628    0.107
## x1           1.93341    0.05243  36.873  < 2e-16 ***
## x2           0.21191    0.04950   4.281 4.37e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9513 on 97 degrees of freedom
## Multiple R-squared:  0.9337, Adjusted R-squared:  0.9323
## F-statistic: 682.8 on 2 and 97 DF,  p-value: < 2.2e-16
```

We note that the results are similar.

We followed all assumptions of Linear Regression in regressing y on x1 and x2 using the test_linear_regression_data data set. We will compare the residual of this regression to that of all the others where assumptions will be broken.

The residual for where all assumptions are met:

```
our_implementation$Residual[1] # a small residual here
```

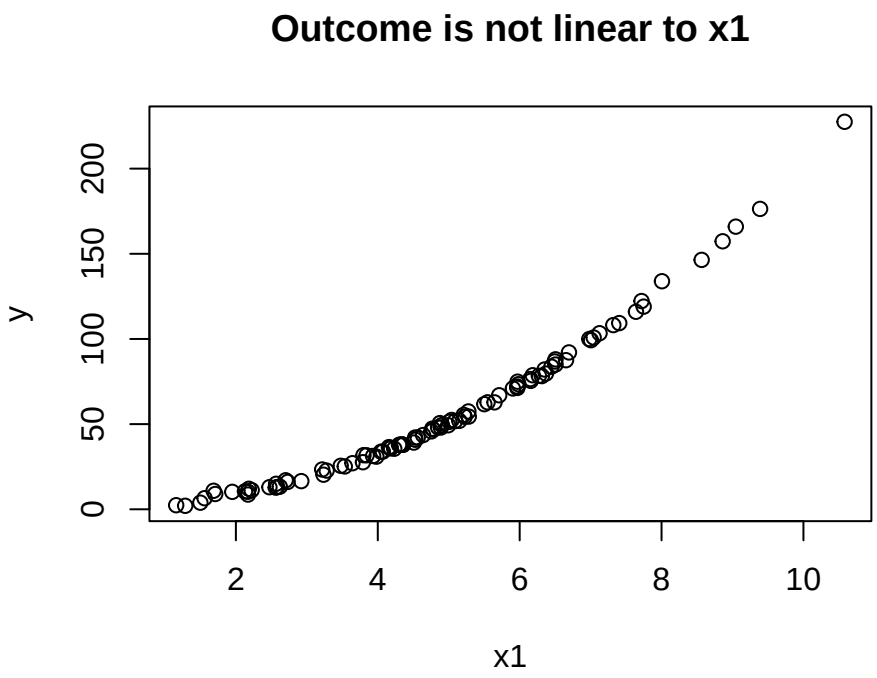
```
## [1] 0.9369198
```

Breaking Assumptions

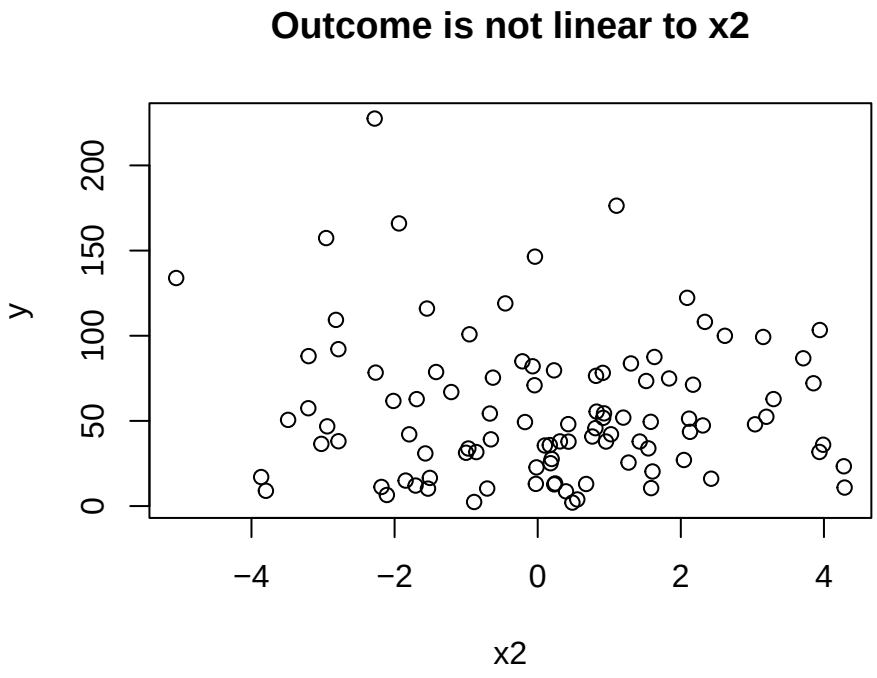
Breaking the assumption of the predictors and outcome following a linear relationship

Creating a data set where, if we apply linear regression, this assumption will be broken.

```
test_linear_regression_data_not_linear <-
  data.frame(
    x1 = rnorm(100, mean = 5, sd = 2),
    x2 = rnorm(100, mean = 0, sd = 2)
  )
error <- rnorm(100, mean = 0, sd = 1)
test_linear_regression_data_not_linear$y <-
  2 * test_linear_regression_data_not_linear$x1^2 + 0.2 *
    test_linear_regression_data_not_linear$x2^2 + error

plot(test_linear_regression_data_not_linear$x1, test_linear_regression_data_not_linear$y,
  xlab = "x1", ylab = "y",
  main = "Outcome is not linear to x1"
)
```



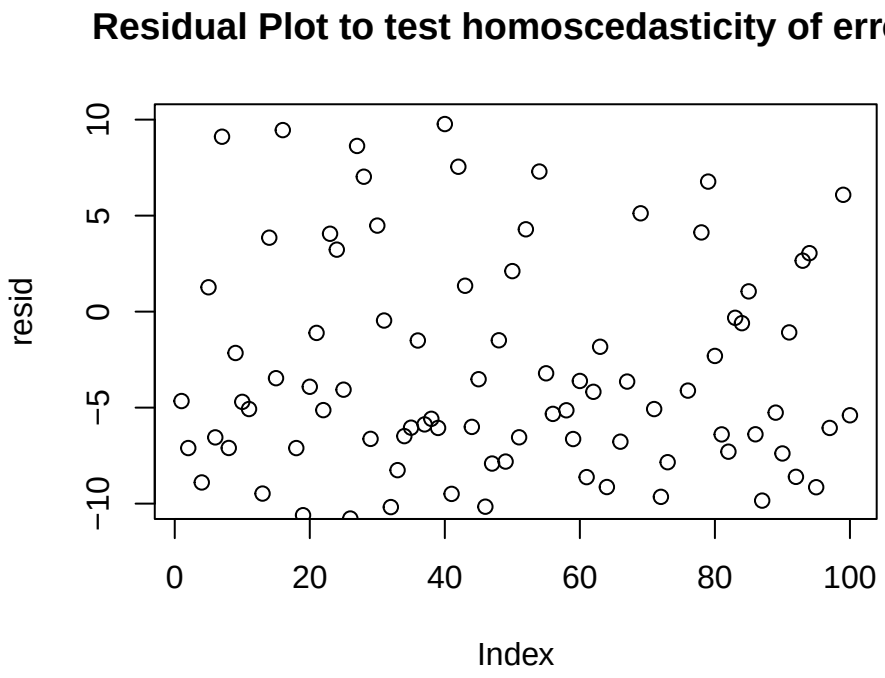

```
plot(test_linear_regression_data_not_linear$x2, test_linear_regression_data_not_linear$y,
  xlab = "x2", ylab = "y",
  main = "Outcome is not linear to x2"
)
```

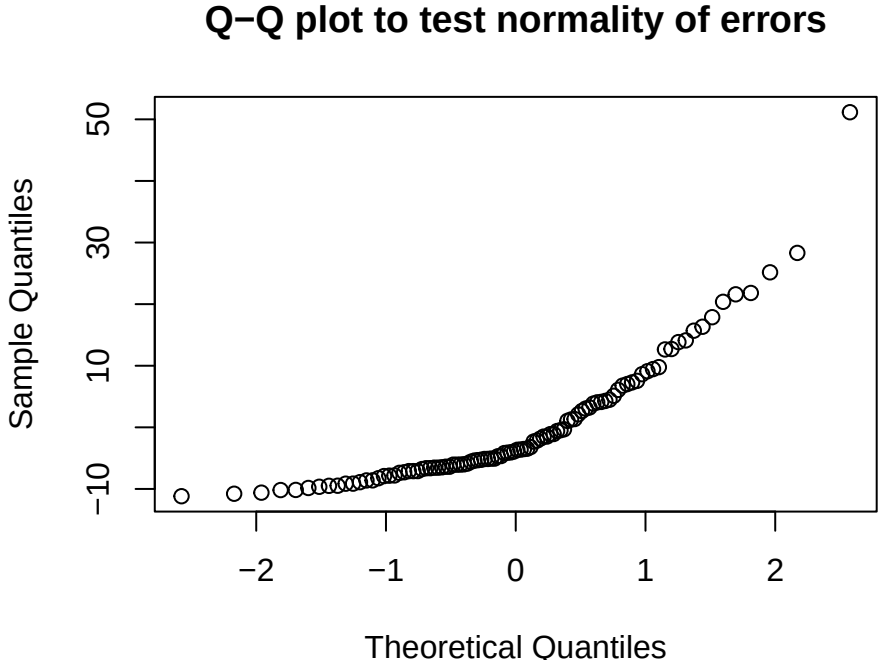


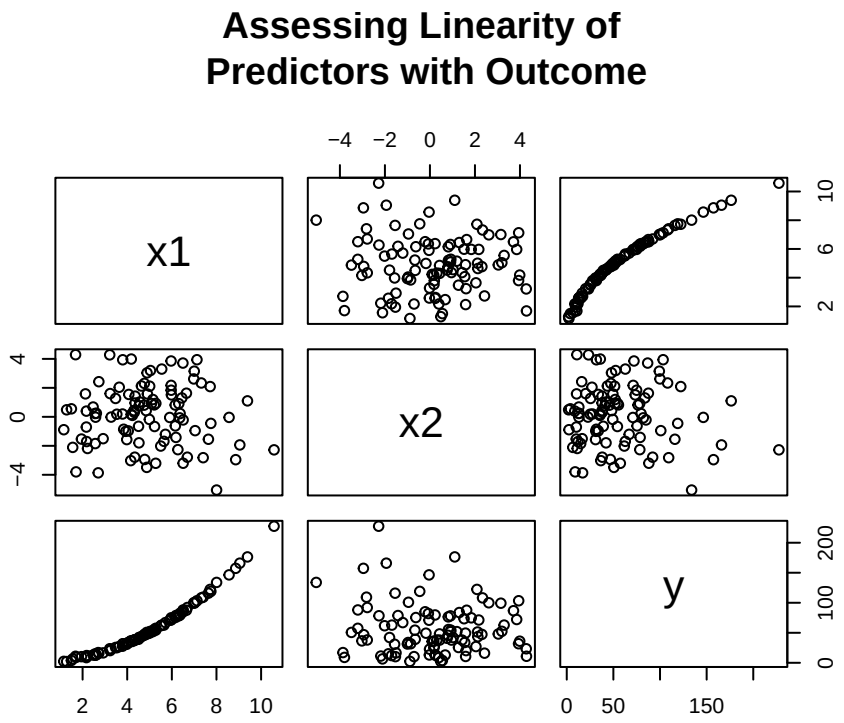
Using our implementation of Linear Regression to fit the model.

```
our_implementation_not_linear <- linear_regression(
  test_linear_regression_data_not_linear,
  test_linear_regression_data_not_linear$x1,
  test_linear_regression_data_not_linear$x2,
  y = test_linear_regression_data_not_linear$y
```

```
)[[2]]
```


Residual Plot to test homoscedasticity of errors
resid
10
5
0
−5
−10
0
20
40
60
80
100
Index


Q−Q plot to test normality of errors
Sample Quantiles
50
30
10
−10
−2
−1
0
1
2
Theoretical Quantiles




```
our_implementation_not_linear$Residual[1] # a higher residual here
```

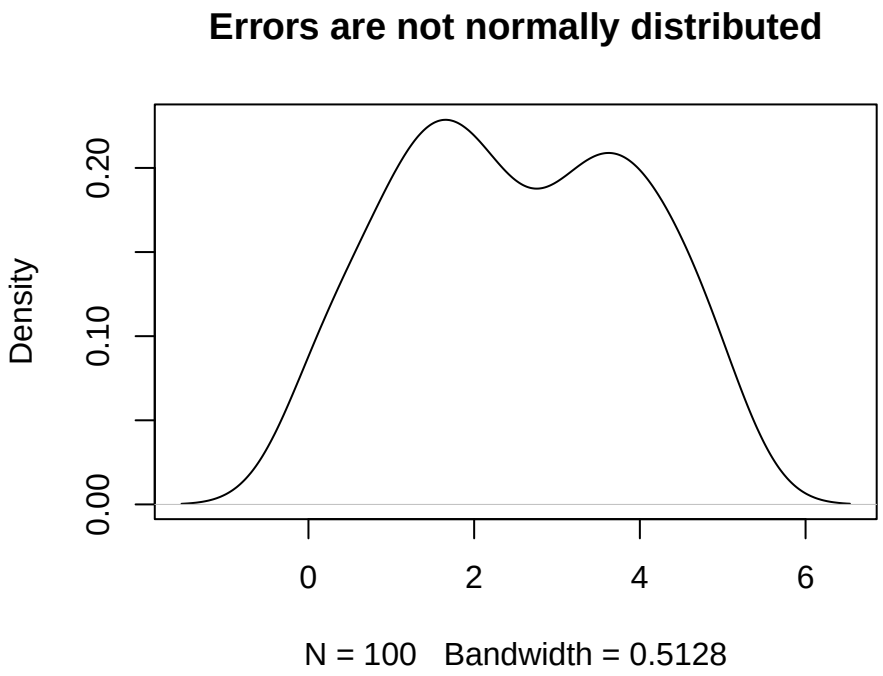
```
## [1] 10.22817
```

We note that linear regression is not performing as well in this case.

Breaking the assumption of the errors being normally distributed

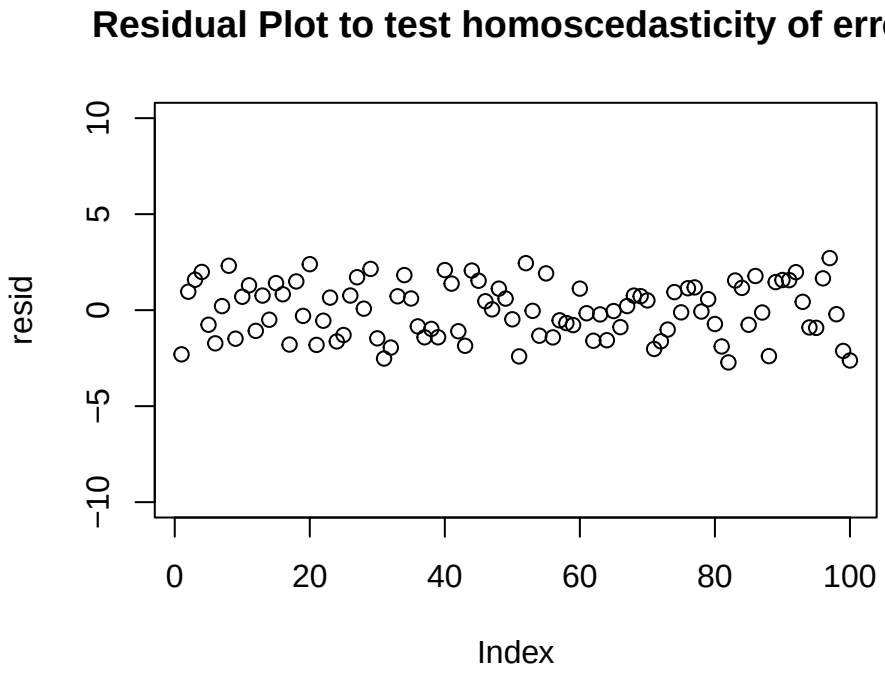
Creating a data set where, if we apply linear regression, this assumption will be broken.

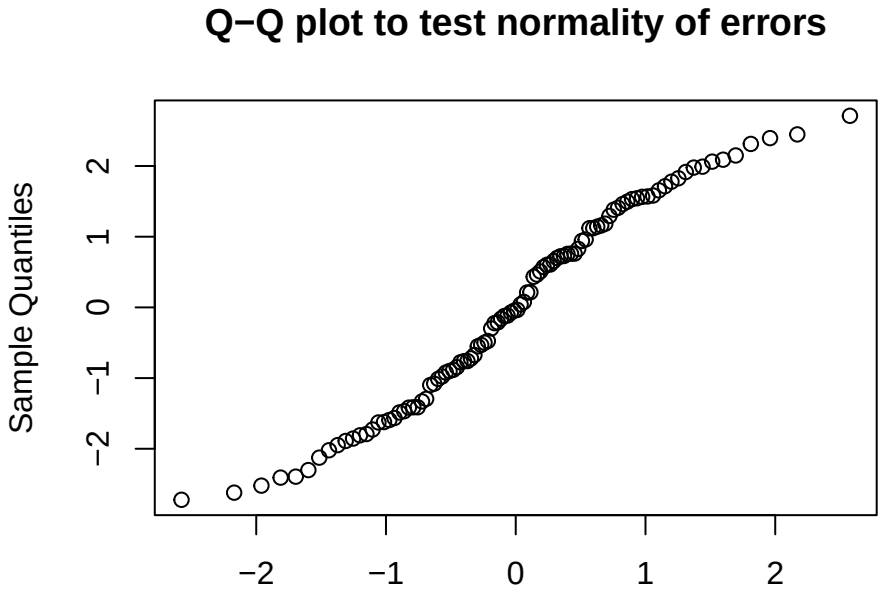
```
test_linear_regression_data_not_normally_dist <-
  data.frame(
    x1 = rnorm(100, mean = 5, sd = 2),
    x2 = rnorm(100, mean = 0, sd = 2)
  )
error <- runif(100, min = 0, max = 5)
test_linear_regression_data_not_normally_dist$y <-
  2 * test_linear_regression_data_not_normally_dist$x1 + 0.2 *
    test_linear_regression_data_not_normally_dist$x2 + error

plot(density(error), main = "Errors are not normally distributed")
```

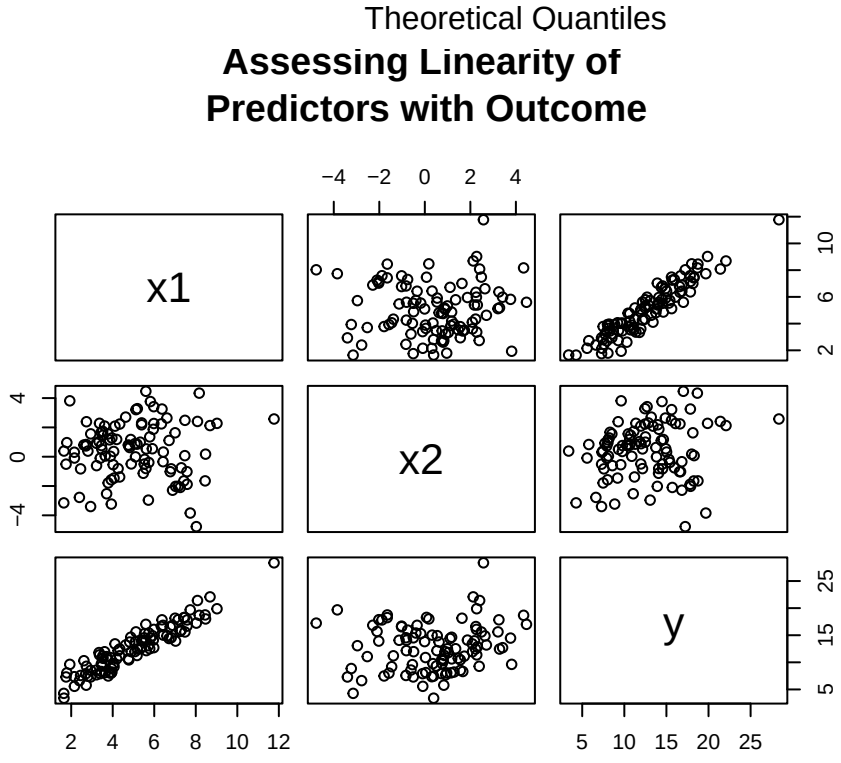


Using our implementation of lm to fit the model.

```
our_implementation_not_normally_dist <- linear_regression(
  test_linear_regression_data_not_normally_dist,
  test_linear_regression_data_not_normally_dist$x1,
  test_linear_regression_data_not_normally_dist$x2,
  y = test_linear_regression_data_not_normally_dist$y
)[[2]]
```








```
our_implementation_not_normally_dist$Residual[1] # a higher residual here
```

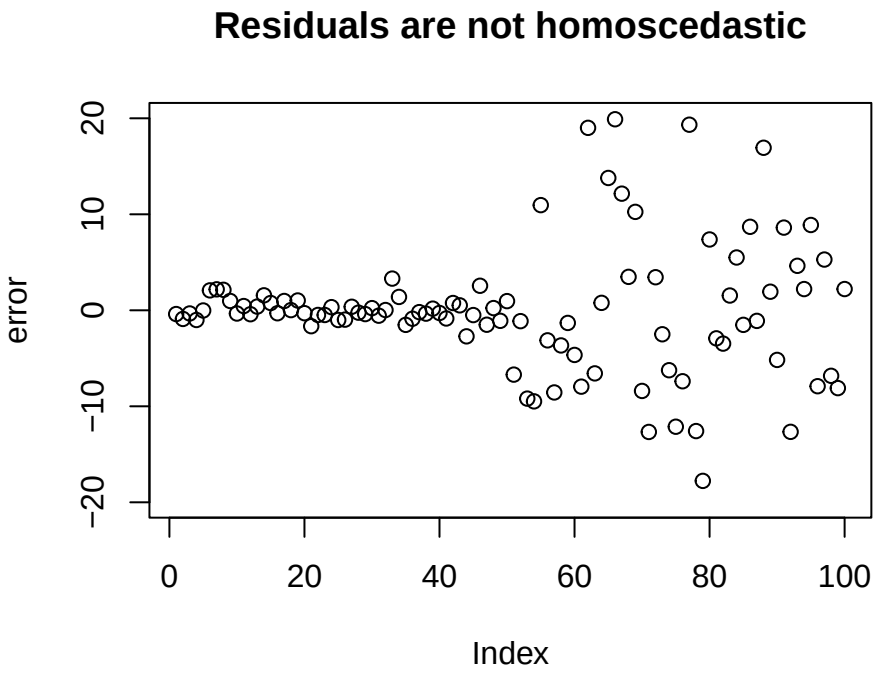
```
## [1] 1.414274
```

We note that linear regression is not performing as well in this case.

Breaking the assumption of the errors being homoscedastic

Creating a data set where, if we apply linear regression, this assumption will be broken.

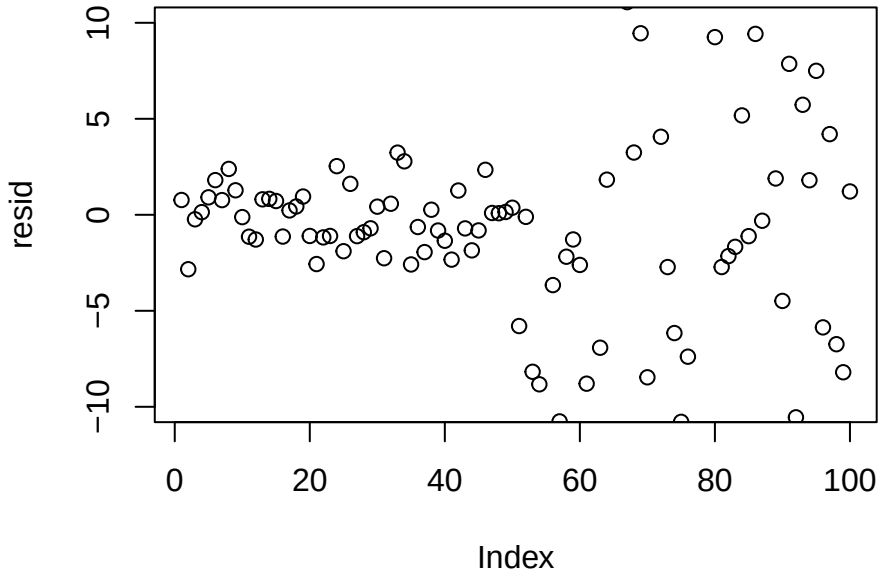
```r
test_linear_regression_data_not_homoscedastic <-
  data.frame(
    x1 = rnorm(100, mean = 5, sd = 2),
    x2 = rnorm(100, mean = 0, sd = 2)
  )
error <- c(
  rnorm(50, mean = 0, sd = 1),
  rnorm(50, mean = 0, sd = 10)
)
test_linear_regression_data_not_homoscedastic$y <-
  2 * test_linear_regression_data_not_homoscedastic$x1 + 0.2 *
    test_linear_regression_data_not_homoscedastic$x2 + error

plot(error,
  ylab = "error", main = "Residuals are not homoscedastic", ylim = c(-20, 20)
)
```



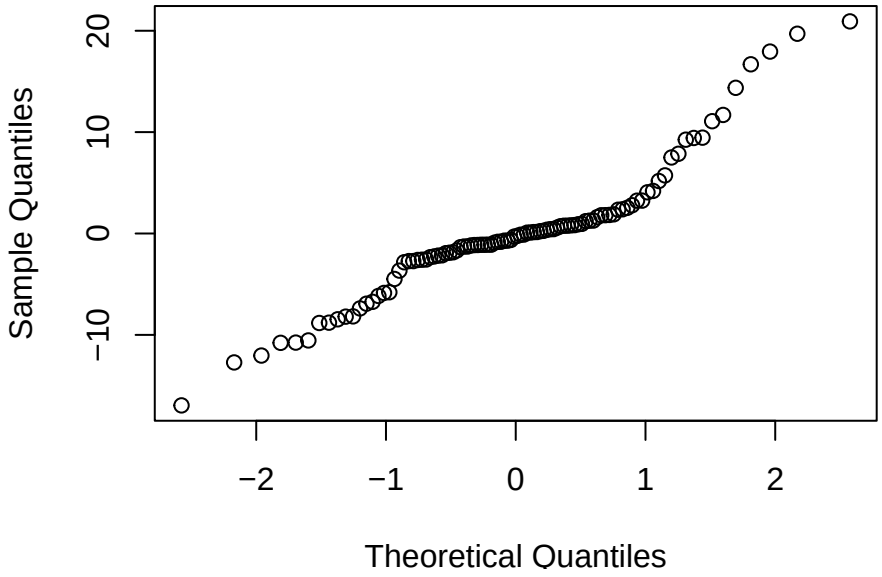
Using our implementation of lm to fit the model.

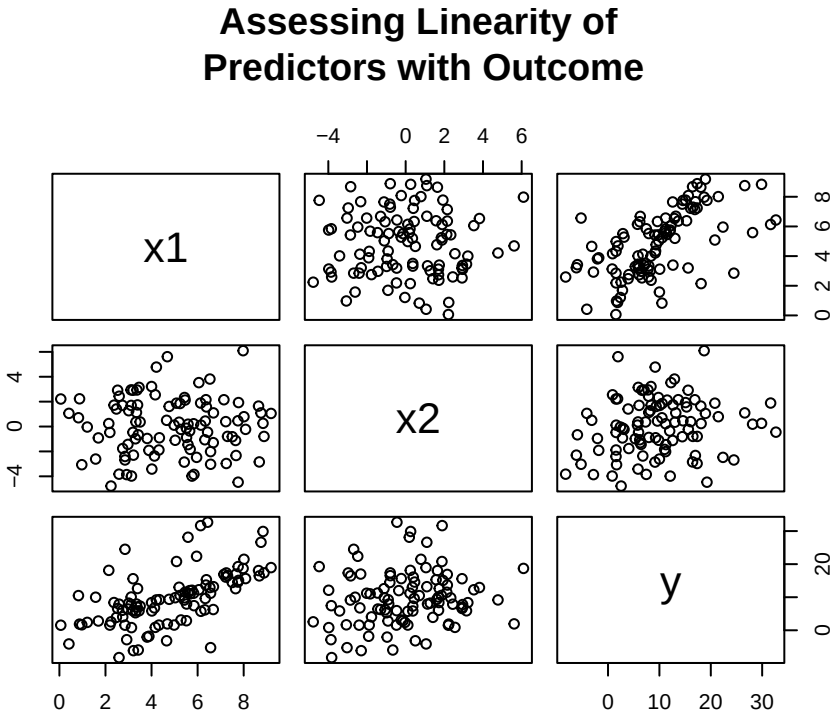
```r
our_implementation_not_homoscedastic <- linear_regression(
  test_linear_regression_data_not_homoscedastic,
  test_linear_regression_data_not_homoscedastic$x1,
  test_linear_regression_data_not_homoscedastic$x2,
  y = test_linear_regression_data_not_homoscedastic$y
)[[2]]
```

Residual Plot to test homoscedasticity of errors



Q−Q plot to test normality of errors






```
our_implementation_not_homoscedastic$Residual[1] # a higher residual here
```

```
## [1] 6.419781
```

We note that linear regression is not performing as well in this case.

Comparing residuals when all assumptions were met versus not

```
residual_comparison <-
  t(
    data.frame(
      resid_all_assumptions_met = our_implementation$Residual[1],
      resid_not_linear = our_implementation_not_linear$Residual[1],
      resid_not_normally_dist = our_implementation_not_normally_dist$Residual[1],
      resid_not_homoscedastic = our_implementation_not_homoscedastic$Residual[1]
    )
  )
row.names(residual_comparison) <- c(
  "All assumptions met",
  "Linearity assumption violated",
  "Normality assumption violated",
  "Homoscedasticity assumption violated"
)
colnames(residual_comparison) <- "Residuals"
residual_comparison
```

```
##                                      Residuals
## All assumptions met                  0.9369198
## Linearity assumption violated       10.2281685
## Normality assumption violated        1.4142737
## Homoscedasticity assumption violated 6.4197814
```

Conclusion

The implementation of Linear Regression where all assumptions are met performs the best; i.e. it gives us predictions which are closest to the true outcome values. From the residual comparison, we also note that applying linear regression to data that aren't linear can be especially worrisome.

Probit Regression

Introduction

The Probit model classifies observations into one of two categories (for simple Probit Regression; multinomial Probit Regression can classify observations into more than two categories) by estimating the probability that an observation with particular characteristics is more likely to fall in one category or another.

Uses

Probit Regression is primarily used when the outcome is binary - thus, it is mainly used for classification problems. When covariates are continuous, there are infinite possible values for the outcome if using Linear Regression; Logistic and Probit Regressions are therefore better than Linear if we need to bound the outcome to 0 and 1.

Logistic Regression and Probit Regressions give almost identical results - they just have different link functions. The decision to chose one over the other is discipline-dependent, and it is said that Logistic Regression is better when one has extreme independent variables (where one particular small or large value will overwhelmingly determine if your outcome is 0 or 1 - overriding the effect of most other variables). However, there is no 'right' answer to this debate.

Assumptions

- The outcome is binary
- The z-score of the outcome and the predictor variables have a linear relationship
- The errors are normally distributed and are independent of one another

Our Probit Regression Implementation

Our Probit Regression implementation: (Note that we use bootstrapping to estimate standard errors)

```
probit_regression <- function(data, ..., y) {
  n <- nrow(data)
  x_parameters <- c(...)
  # defining the predictor matrix
  X <-
    matrix(c(rep(1, n), x_parameters),
      nrow = n,
      ncol = ncol(data)
    )
  # defining the outcome matrix
  Y <- matrix(y, nrow = n, ncol = 1)
  # defining the log likelihood
  probit.loglikelihood <- function(beta, X, Y) {
    eta <- X %*% beta
    p <- pnorm(eta)
    loglikelihood <- -sum((1 - Y) * log(1 - p) + Y * log(p))
    return(loglikelihood)
  }
```

```r
# starting with an initial guess of the parameter values
initial_guess <- matrix(0, nrow = ncol(data), ncol = 1)
# using 'optim' to maximize the log likelihood
result <- optim(
  initial_guess,
  fn = probit.loglikelihood,
  X = X,
  Y = Y,
  method = NULL
)$par
# creating a vector 'estimate' for the beta coefficients
estimate <- result
# bootstrapping to estimate the standard errors
num_bootstraps <- 10
result_bootstrap <-
  matrix(0, nrow = num_bootstraps, ncol = ncol(X))
for (i in 1:num_bootstraps) {
  sample_indices <- sample(nrow(data), replace = TRUE)
  bootstrap_data <- data[sample_indices, ]
  X_bootstrap <-
    matrix(
      c(rep(1, nrow(bootstrap_data)), x_parameters),
      nrow = nrow(bootstrap_data),
      ncol = ncol(bootstrap_data)
    )
  Y_bootstrap <-
    matrix(bootstrap_data$y,
      nrow = nrow(bootstrap_data),
      ncol = 1
    )
  initial_guess_bootstrap <-
    matrix(0, nrow = ncol(bootstrap_data), ncol = 1)
  result_bootstrap[i, ] <- optim(
    initial_guess_bootstrap,
    probit.loglikelihood,
    X = X_bootstrap,
    Y = Y_bootstrap,
    method = NULL
  )$par
}
# finding the standard deviation of the bootstrapped betas to find the
# standard error of the coefficients
se <- apply(result_bootstrap, 2, sd)
# calculating the z-statistic
z <- estimate / se
# defining the degrees of freedom
df <- nrow(X) - ncol(X)
# calculating the p-value
p <- 2 * pnorm(z, lower.tail = FALSE)
# defining the row names of the output data frame
rownames <- c()
for (i in 1:((ncol(X)) - 1)) {
  rownames[i] <- i
```

```
  }
  data_to_plot <- data[, -which(colnames(data) == "y")]
  data_to_plot$y_zscore <- qnorm(pnorm(data$y))
  test <- list(
    pairs(data_to_plot, main = "Assessing Linearity of Predictors\n with z score of Outcome")
  )
  impl <- data.frame(
    Estimate = estimate,
    Std.Error = se,
    z.value = z,
    p.value = p,
    DegOfFreedom = c(df, rep(NA, ncol(X) - 1)),
    row.names = c("(Intercept)", paste0(rep("x", ncol(
      X
    ) - 1), rownames))
  )
  # returning a data frame akin to the glm probit output
  return(list(test, impl))
}
```

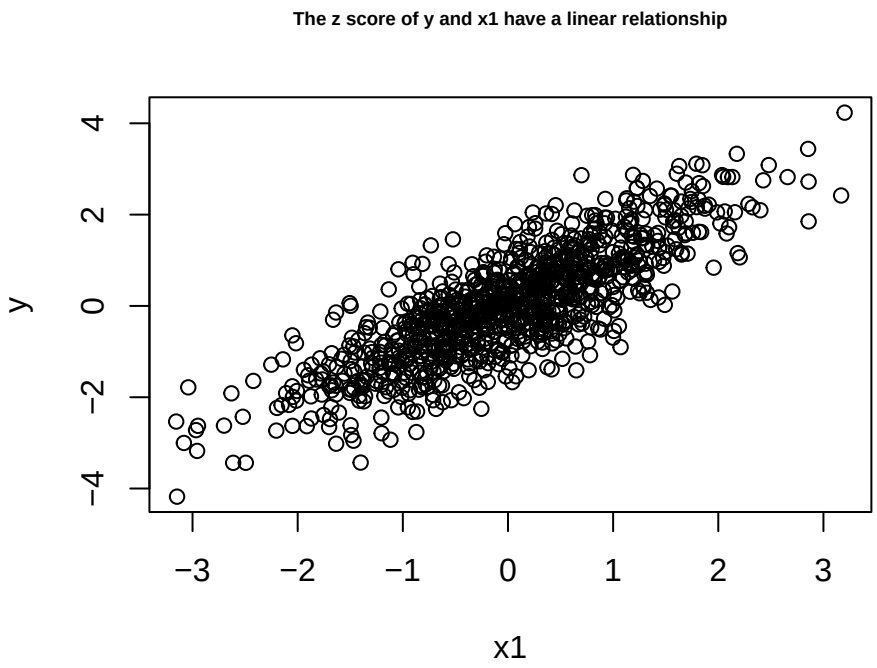
Creating a function to predict the outcomes based on our Probit Regression implementation.

```
predict_probit <-
  function(data, ..., y, implementation_probit) {
    n <-
      implementation_probit$DegOfFreedom[1] + nrow(implementation_probit)
    input_covariate_values <- c(...)
    X <-
      matrix(
        c(rep(1, n), input_covariate_values),
        nrow = n,
        ncol = nrow(implementation_probit)
      )
    Y <- matrix(y, nrow = n, ncol = 1)
    estimate <-
      implementation_probit[1:nrow(implementation_probit), 1]
    pred <- ifelse(X %*% estimate < 0, 0, 1)
    return(pred)
  }
```

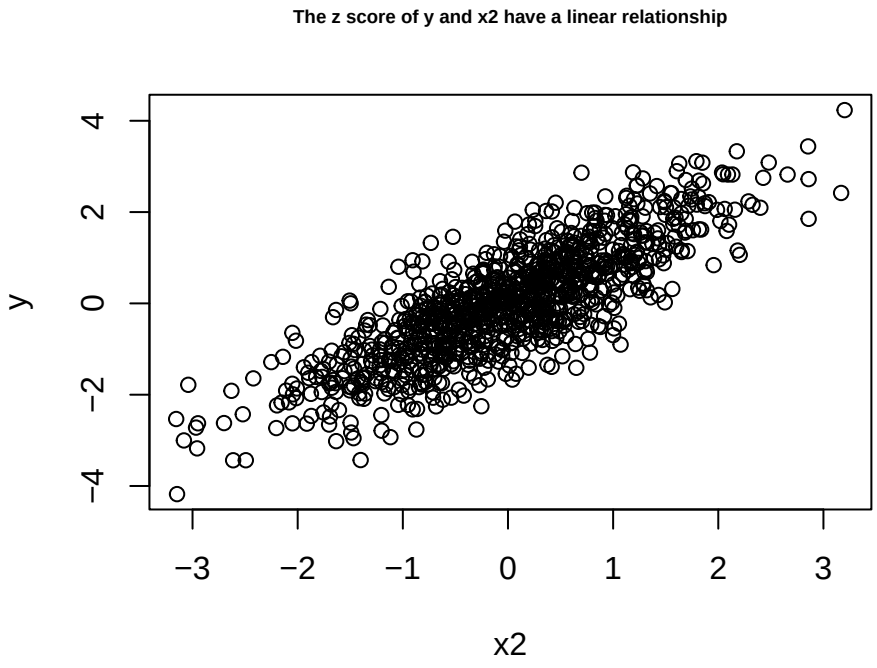
Creating a test data set which meets all Probit Regression assumptions to check if our function works.

```
test_probit_regression_data <- data.frame(
  x1 = rnorm(1000, 0, 1),
  x2 = rnorm(1000, 0, 1)
)
error <- rnorm(1000, mean = 0, sd = 0.5)
test_probit_regression_data$y <- test_probit_regression_data$x1 +
  0.5 * test_probit_regression_data$x2 +
  error
test_probit_regression_data$y <-
  qnorm(pnorm(test_probit_regression_data$y))

plot(test_probit_regression_data$x1, test_probit_regression_data$y,
  main = "The z score of y and x1 have a linear relationship", cex.main = 0.6,
```

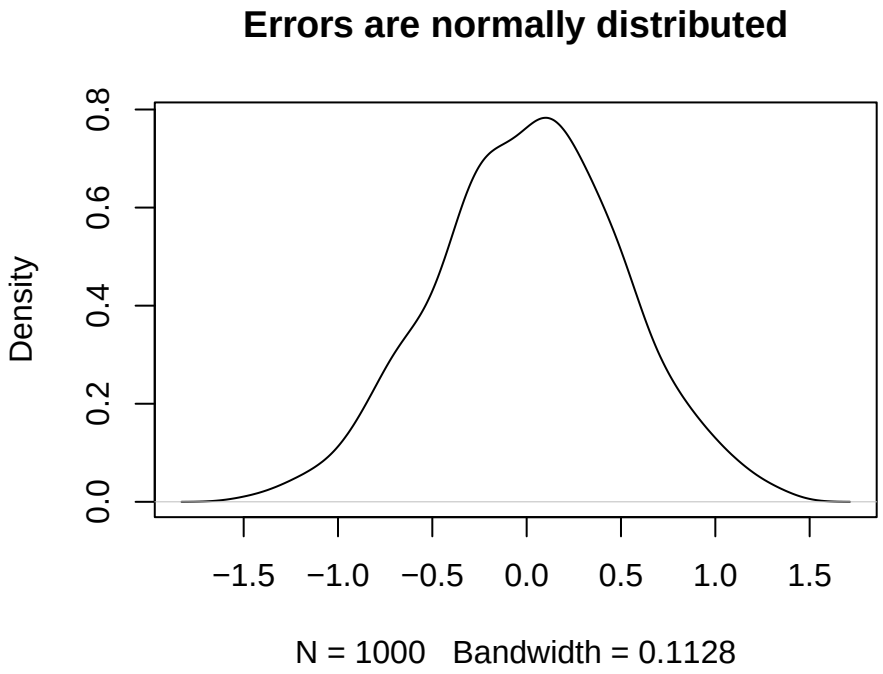
```
  xlab = "x1", ylab = "y"
)
```




```
plot(test_probit_regression_data$x1, test_probit_regression_data$y,
  main = "The z score of y and x2 have a linear relationship", cex.main = 0.6,
  xlab = "x2", ylab = "y"
)
```

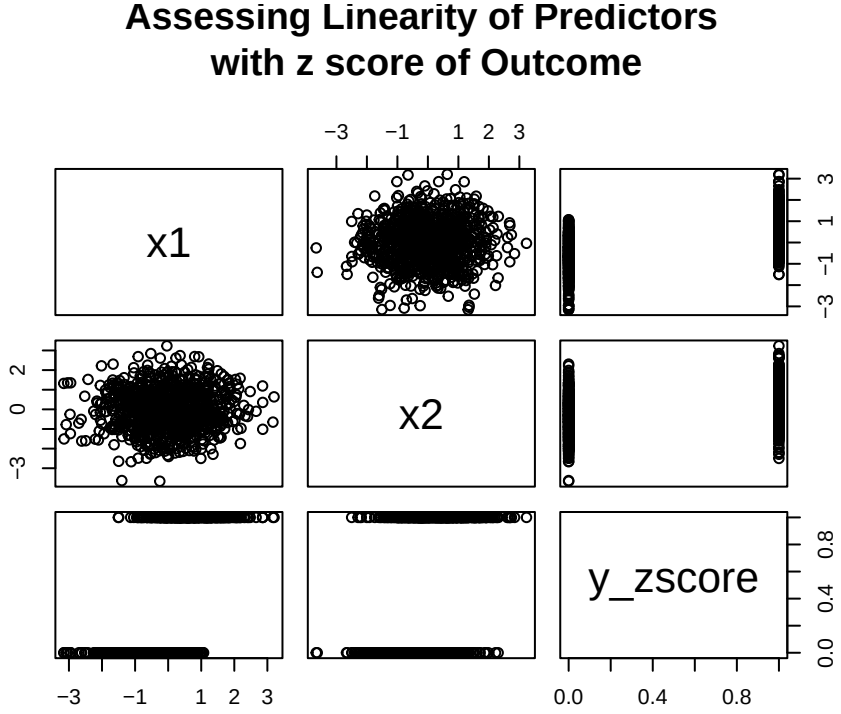



```
test_probit_regression_data$y <-
  ifelse(test_probit_regression_data$y < 0, 0, 1)

plot(density(error), main = "Errors are normally distributed")
```



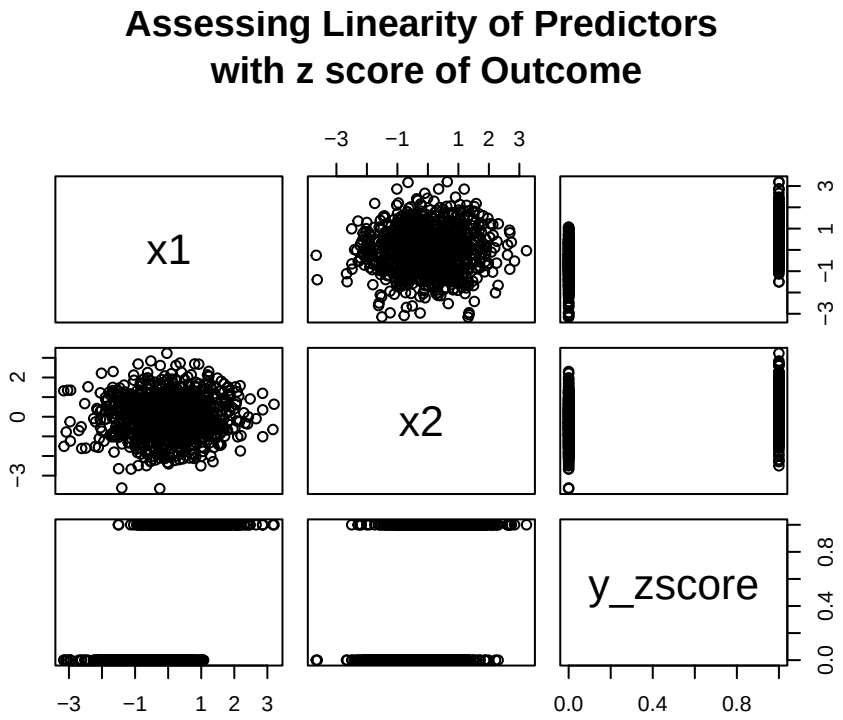
Testing Assumptions for Probit Regression

```
test_probit_reg <- probit_regression(test_probit_regression_data,
  test_probit_regression_data$x1,
  test_probit_regression_data$x2,
  y = test_probit_regression_data$y
)[[1]]
```



Applying the function we created on this data set.

```
our_implementation_probit <-
  probit_regression(
    test_probit_regression_data,
    test_probit_regression_data$x1,
    test_probit_regression_data$x2,
    y = test_probit_regression_data$y
  )[[2]]
```




```
our_implementation_probit
```

```
##                Estimate  Std.Error    z.value       p.value DegOfFreedom
## (Intercept) 0.008642326 0.02426245  0.3562017  7.216895e-01          997
## x1          1.861391305 0.04321815 43.0696708  0.000000e+00           NA
## x2          0.944419878 0.03281213 28.7826480 3.537093e-182           NA
```

Comparing our output to R's output.

```
r_implementation_probit <-
  summary(glm(y ~ x1 + x2,
    data = test_probit_regression_data,
    family = binomial(link = "probit")
  ))
r_implementation_probit
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = binomial(link = "probit"),
##     data = test_probit_regression_data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.64955  -0.46896  -0.00002   0.43274   2.86456
##
```

```
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 0.008606   0.056552   0.152    0.879
## x1          1.861287   0.112248  16.582   <2e-16 ***
## x2          0.944549   0.077920  12.122   <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: 1386.23  on 999  degrees of freedom
## Residual deviance:  638.38  on 997  degrees of freedom
## AIC: 644.38
##
## Number of Fisher Scoring iterations: 7
```

We note that the results are similar.

We followed all assumptions of Probit Regression in regressing y on x1 and x2 using the test_probit_regression_data data set. We will compare the residual of this regression to that of all the others where assumptions will be broken.

The accuracy for where all assumptions are met:

```
prediction_all_assumptions_met <-
  as.numeric(
    predict_probit(
      test_probit_regression_data,
      test_probit_regression_data$x1,
      test_probit_regression_data$x2,
      y = test_probit_regression_data$y,
      implementation_probit = our_implementation_probit
    )
  )
accuracy_all_assumptions_met <-
  sum(prediction_all_assumptions_met == test_probit_regression_data$y) / 1000
accuracy_all_assumptions_met # high accuracy here
```

```
## [1] 0.852
```

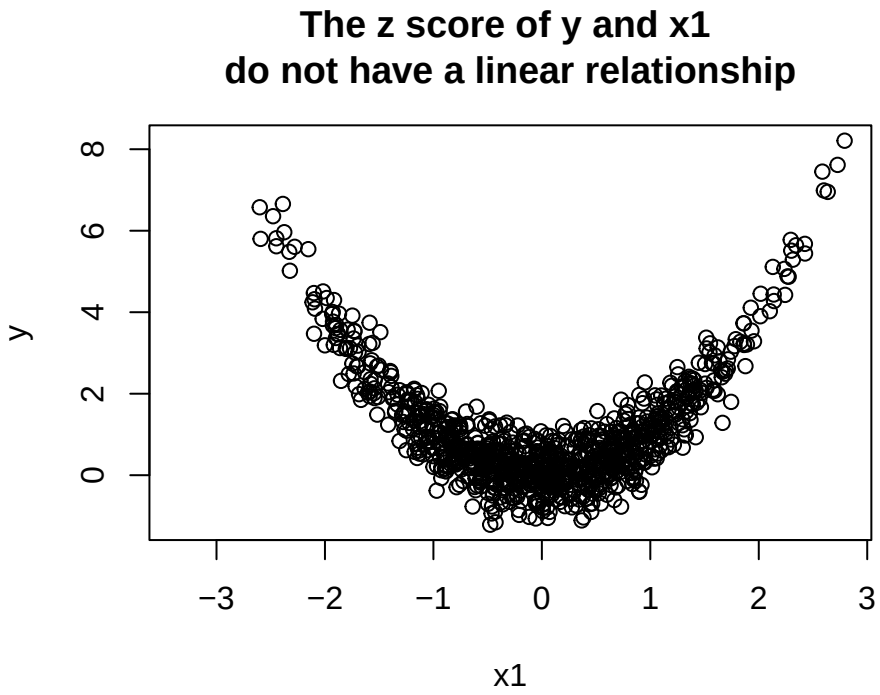
Breaking Assumptions

Breaking the assumption that the relationship between the predictors and the z score of y is linear

Creating a data set where, if we apply Probit Regression, this assumption will be broken.

```
test_probit_regression_data_not_linear <-
  data.frame(
    x1 = rnorm(1000, 0, 1),
    x2 = rnorm(1000, 0, 1)
  )
eror <- rnorm(1000, mean = 0, sd = 0.5)
test_probit_regression_data_not_linear$y <-
  test_probit_regression_data_not_linear$x1^2 + error
test_probit_regression_data_not_linear$y <-
  qnorm(pnorm(test_probit_regression_data_not_linear$y))
```

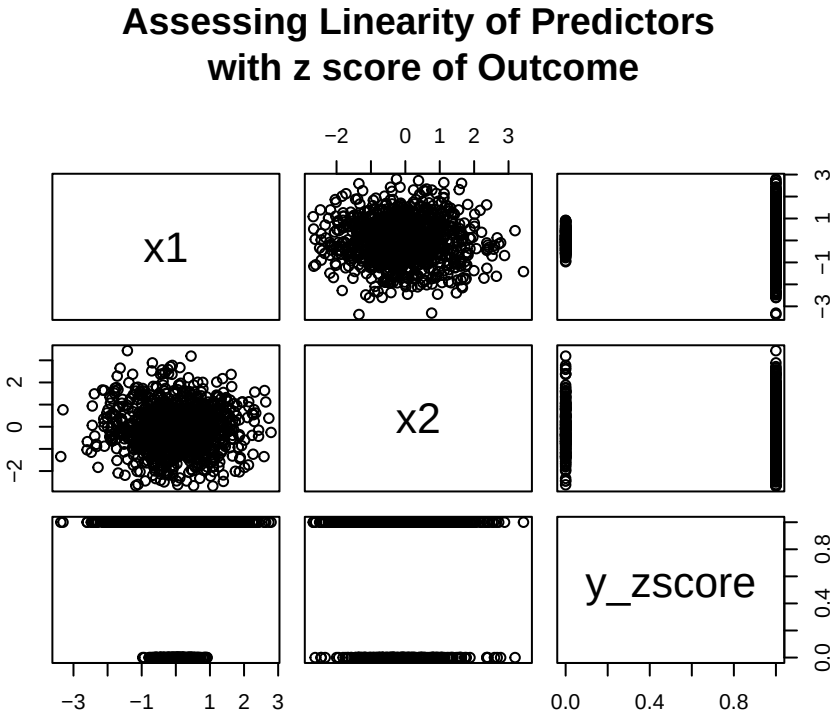
```r
plot(test_probit_regression_data_not_linear$x1, test_probit_regression_data_not_linear$y,
  main = "The z score of y and x1 \ndo not have a linear relationship",
  xlab = "x1", ylab = "y"
)
```




```r
test_probit_regression_data_not_linear$y <-
  ifelse(test_probit_regression_data_not_linear$y < 0, 0, 1)
```

Using our implementation of glm Probit to fit the model and get an accuracy measure.

```r
our_implementation_probit_not_linear <-
  probit_regression(
    test_probit_regression_data_not_linear,
    test_probit_regression_data_not_linear$x1,
    test_probit_regression_data_not_linear$x2,
    y = test_probit_regression_data_not_linear$y
  )[[2]]
```




```
prediction_not_linear <-
  as.numeric(
    predict_probit(
      test_probit_regression_data_not_linear,
      test_probit_regression_data_not_linear$x1,
      test_probit_regression_data_not_linear$x2,
      y = test_probit_regression_data_not_linear$y,
      implementation_probit = our_implementation_probit_not_linear
    )
  )
accuracy_not_linear <-
  sum(prediction_not_linear == test_probit_regression_data_not_linear$y) / 1000
accuracy_not_linear # lower accuracy here
```

```
## [1] 0.797
```

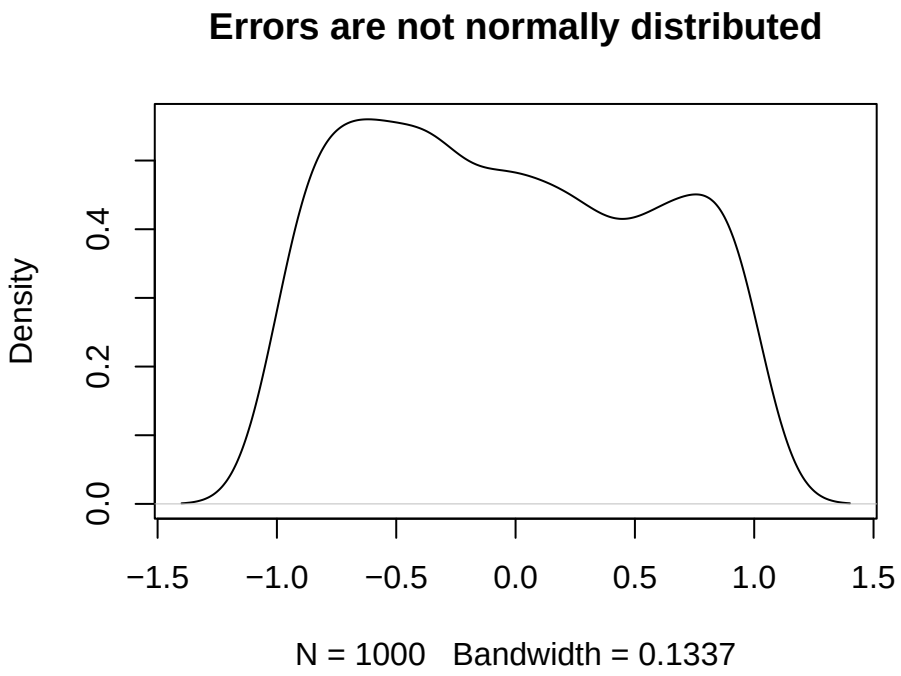
We note that Probit Regression is not performing as well in this case.

Breaking the assumption that the errors are normally distributed

Creating a data set where, if we apply Probit Regression, this assumption will be broken.

```
test_probit_regression_data_not_normally_dist <-
  data.frame(
    x1 = rnorm(1000, 0, 1),
    x2 = rnorm(1000, 0, 1)
  )
error <- runif(1000, min = -1, max = 1)
test_probit_regression_data_not_normally_dist$y <-
  test_probit_regression_data_not_normally_dist$x1 + error
test_probit_regression_data_not_normally_dist$y <-
  qnorm(pnorm(test_probit_regression_data_not_normally_dist$y))
```

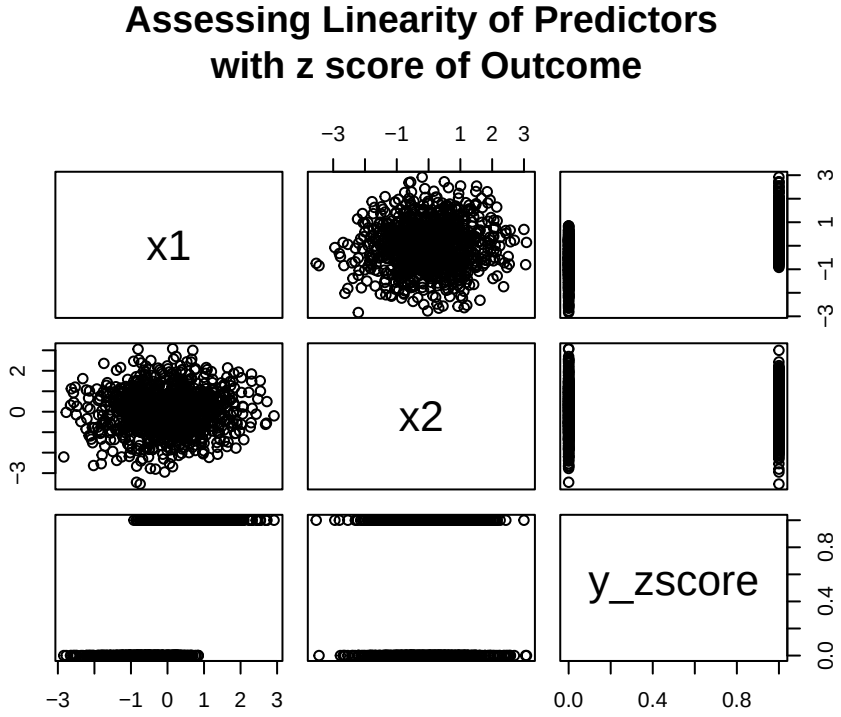
```r
plot(density(error), main = "Errors are not normally distributed")
```




```r
test_probit_regression_data_not_normally_dist$y <-
  ifelse(test_probit_regression_data_not_normally_dist$y < 0, 0, 1)
```

Using our implementation of glm Probit to fit the model and get an accuracy measure.

```r
our_implementation_probit_not_normally_dist <-
  probit_regression(
    test_probit_regression_data_not_normally_dist,
    test_probit_regression_data_not_normally_dist$x1,
    test_probit_regression_data_not_normally_dist$x2,
    y = test_probit_regression_data_not_normally_dist$y
  )[[2]]
```




```
prediction_not_normally_dist <-
  as.numeric(
    predict_probit(
      test_probit_regression_data_not_normally_dist,
      test_probit_regression_data_not_normally_dist$x1,
      test_probit_regression_data_not_normally_dist$x2,
      y = test_probit_regression_data_not_normally_dist$y,
      implementation_probit = our_implementation_probit_not_normally_dist
    )
  )
accuracy_not_normally_dist <-
  sum(prediction_not_normally_dist == test_probit_regression_data_not_normally_dist$y) / 1000
accuracy_not_normally_dist # lower accuracy here
```

```
## [1] 0.804
```

We note that Probit Regression is not performing as well in this case.

Comparing accuracies when all assumptions were met versus not

```
accuracy_comparison <-
  t(
    data.frame(
      accuracy_all_assumptions_met,
      accuracy_not_linear,
      accuracy_not_normally_dist
    )
  )
row.names(accuracy_comparison) <- c(
  "All assumptions met",
  "Linearity assumption violated",
  "Normality assumption violated"
```

```
)
colnames(accuracy_comparison) <- "Accuracy"
accuracy_comparison
```

```
##                                Accuracy
## All assumptions met               0.852
## Linearity assumption violated     0.797
## Normality assumption violated     0.804
```

Conclusion

The implementation of Probit Regression where all assumptions are met performs the best; i.e. it gives us predictions which are more accurate to the true outcome values.

Negative Binomial Regression

Introduction

Negative Binomial Regression is used for predicting count data, similar to Poisson Regression, but the Negative Binomial is more flexible as it allows for the variance of the outcome to be greater than its mean (in Poisson Regression, they are assumed to be equal).

Uses

Negative Binomial Regression is used to model count data with excess zeros (as in the Zero-Inflated Negative Binomial Regression) and is used to model rare events which are less likely to have counts where mean = variance. Negative Binomial can be extended to handle correlated/clustered data as well.

Assumptions

- The outcome represents count data
- The variance of the outcome is greater than its mean
- The relationship between the predictors and the log of the outcome's mean is linear
- The errors are independent of one another

Our Negative Binomial Regression Implementation

Our Negative Binomial Regression implementation: (Note that we use bootstrapping to estimate standard errors)

```
negative_binomial_regression <- function(data, ..., y) {
  n <- nrow(data)
  x_parameters <- c(...)
  # defining the predictor matrix
  X <-
    matrix(c(rep(1, n), x_parameters),
      nrow = n,
      ncol = ncol(data)
    )
  # defining the outcome matrix
  Y <- matrix(y, nrow = n, ncol = 1)
  # starting with theta = 1
  theta <- 1
  # defining the log likelihood
```

```r
negative_binomial.likelihood <- function(beta, X, Y = y) {
  eta <- X %*% beta
  mu <- exp(eta)
  loglikelihood <-
    sum(Y * log(mu) - (Y + 1 / theta) * log(1 + mu / theta))
  return(loglikelihood)
}
# starting with an initial guess of the parameter values
initial_guess <- rep(0, ncol(X))
# using 'optim' to maximize the log likelihood
result <- optim(
  initial_guess,
  negative_binomial.likelihood,
  X = X,
  Y = Y,
  control = list(fnscale = -1),
  hessian = T,
  method = NULL
)$par
# creating a vector 'estimate' for the beta coefficients
estimate <- result
# bootstrapping to estimate the standard errors
num_bootstraps <- 10
result_bootstrap <-
  matrix(0, nrow = num_bootstraps, ncol = ncol(X))
for (i in 1:num_bootstraps) {
  sample_indices <- sample(nrow(data), replace = TRUE)
  bootstrap_data <- data[sample_indices, ]
  X_bootstrap <-
    matrix(
      c(rep(1, nrow(bootstrap_data)), x_parameters),
      nrow = nrow(bootstrap_data),
      ncol = ncol(bootstrap_data)
    )
  Y_bootstrap <-
    matrix(bootstrap_data$y,
      nrow = nrow(bootstrap_data),
      ncol = 1
    )
  initial_guess_bootstrap <-
    matrix(0, nrow = ncol(bootstrap_data), ncol = 1)
  result_bootstrap[i, ] <- optim(
    initial_guess_bootstrap,
    negative_binomial.likelihood,
    X = X_bootstrap,
    Y = Y_bootstrap,
    control = list(fnscale = -1),
    hessian = T,
    method = NULL
  )$par
}
# finding the standard deviation of the bootstrapped betas to find the
# standard error of the coefficients
```

```
  se <- apply(result_bootstrap, 2, sd)
  # calculating the z-statistic
  z <- estimate / se
  # defining the degrees of freedom
  df <- nrow(X) - ncol(X)
  # calculating the p-value
  p <- 2 * pnorm(z, lower.tail = FALSE)
  # defining the row names of the output data frame
  rownames <- c()
  for (i in 1:((ncol(X)) - 1)) {
    rownames[i] <- i
  }
  data_to_plot <- data[, -which(colnames(data) == "y")]
  data_to_plot$y_log <- log(data$y)
  test <- list(
    pairs(data_to_plot, main = "Assessing Linearity of Predictors \nwith log of Outcome")
  )
  impl <- data.frame(
    Estimate = estimate,
    Std.Error = se,
    z.value = z,
    p.value = p,
    DegOfFreedom = c(df, rep(NA, ncol(X) - 1)),
    row.names = c("(Intercept)", paste0(rep("x", ncol(
      X
    ) - 1), rownames))
  )
  # returning a data frame akin to the glm probit output
  return(list(test, impl))
}
```

Creating a function to predict the outcomes based on our Negative Binomial Regression implementation.

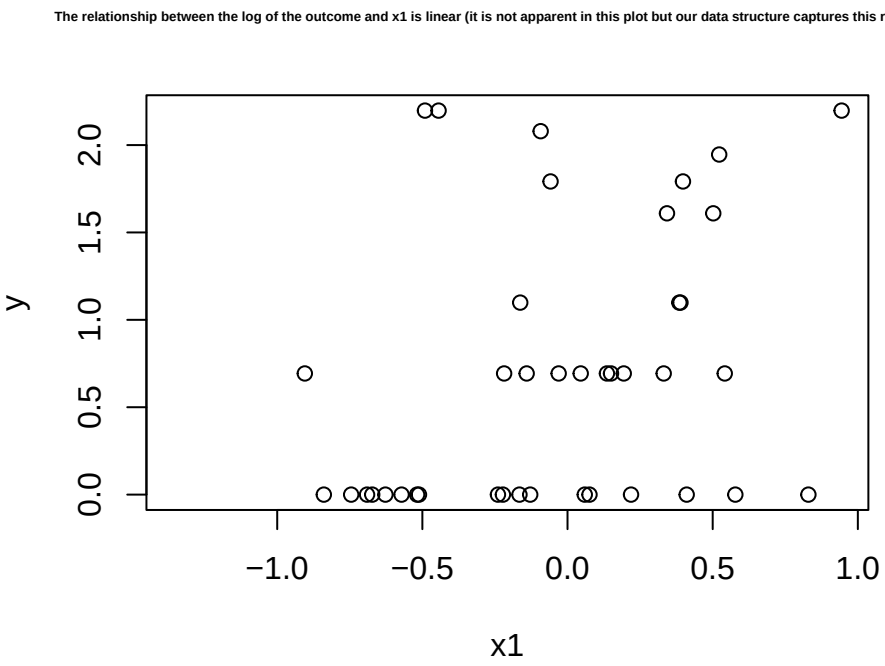
```
predict_neg_binom <-
  function(data, ..., y, implementation_neg_binom) {
    n <-
      implementation_neg_binom$DegOfFreedom[1] + nrow(implementation_neg_binom)
    input_covariate_values <- c(...)
    X <-
      matrix(
        c(rep(1, n), input_covariate_values),
        nrow = n,
        ncol = nrow(implementation_neg_binom)
      )
    Y <- matrix(y, nrow = n, ncol = 1)
    estimate <-
      implementation_neg_binom[1:nrow(implementation_neg_binom), 1]
    pred <- exp(X %*% estimate)
    return(pred)
  }
```

Creating a test data set which meets all Negative Binomial Regression assumptions to check if our function works.

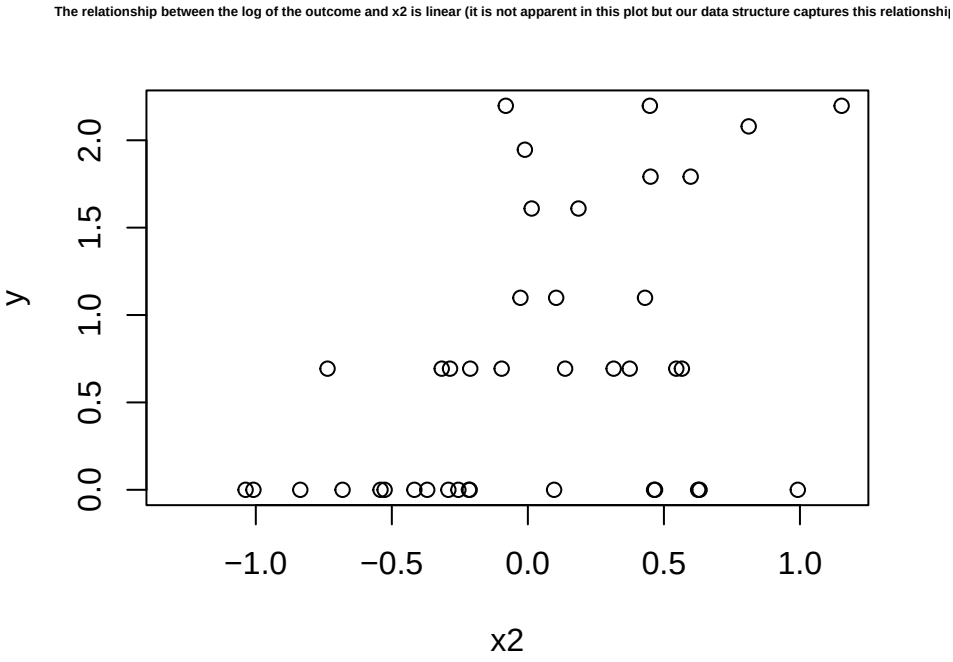
```r
x1 <- rnorm(100, mean = 0, sd = 0.5)
x2 <- rnorm(100, mean = 0, sd = 0.5)
y <- rnbinom(100, mu = exp(x1 + x2), size = 0.5)
test_neg_binom_regression_data <- data.frame(x1, x2, y)
# to ensure that the variance of the outcome variable is greater
# than its mean
var(y) > mean(y)
```

```
## [1] TRUE
```

```r
plot(test_neg_binom_regression_data$x1, log(test_neg_binom_regression_data$y),
  main = "The relationship between the log of the outcome and x1 is linear (it is not apparent in this p
  xlab = "x1", ylab = "y"
)
```

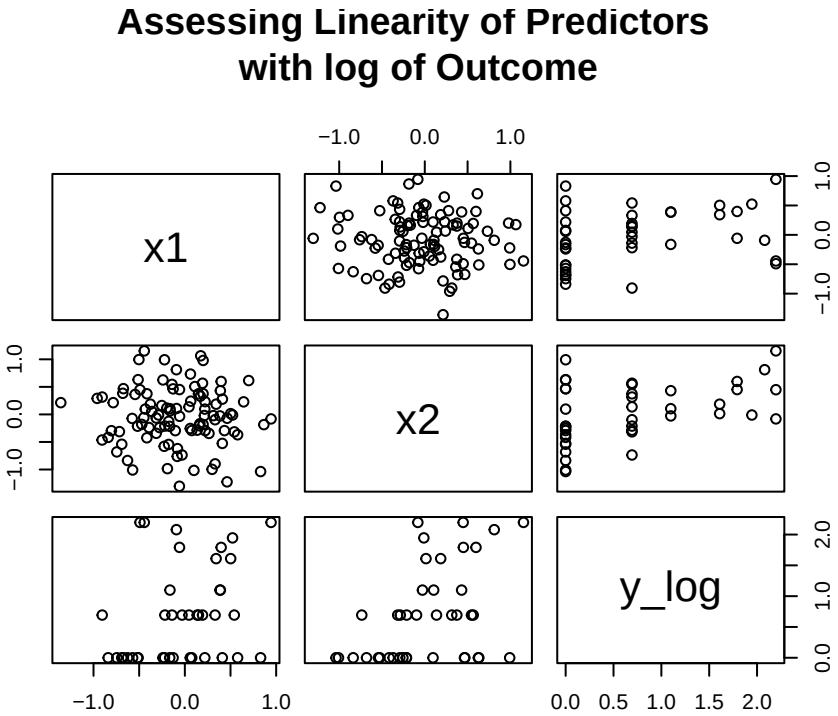



```r
plot(test_neg_binom_regression_data$x2, log(test_neg_binom_regression_data$y),
  xlab = "x2", ylab = "y",
  main = "The relationship between the log of the outcome and x2 is linear (it is not apparent in this p
)
```



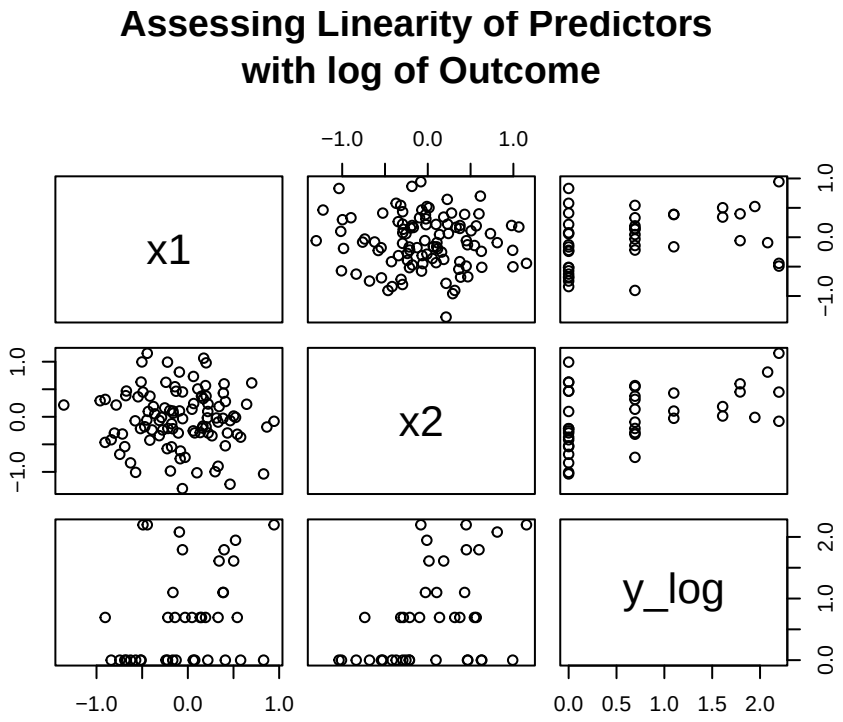
Testing Assumptions for Negative Binomial Regression

```
test_negbinom_reg <- negative_binomial_regression(test_neg_binom_regression_data,
  test_neg_binom_regression_data$x1,
  test_neg_binom_regression_data$x2,
  y = test_neg_binom_regression_data$y
)[[1]]
```



Using our implementation of Negative Binomial to fit the model and get residual measure.

```
our_implementation_neg_binom <-
  negative_binomial_regression(
    test_neg_binom_regression_data,
    test_neg_binom_regression_data$x1,
    test_neg_binom_regression_data$x2,
    y = test_neg_binom_regression_data$y
  )[[2]]
```




```
our_implementation_neg_binom
```

```
##                Estimate Std.Error    z.value     p.value DegOfFreedom
## (Intercept) -0.03119733 0.1773178 -0.1759402 1.139659098           97
## x1           0.95294799 0.3302106  2.8858792 0.003903220           NA
## x2           1.23129249 0.4058991  3.0334937 0.002417398           NA
```

Comparing our output to R's output.

```
r_implementation_neg_binom <-
  summary(glm.nb(y ~ x1 + x2, data = test_neg_binom_regression_data))
r_implementation_neg_binom
```

```
##
## Call:
## glm.nb(formula = y ~ x1 + x2, data = test_neg_binom_regression_data,
##     init.theta = 0.5173150957, link = log)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5062  -1.0168  -0.7598   0.2771   2.0691
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.03126    0.17976  -0.174 0.861936
```

```
## x1           0.91908    0.41227   2.229 0.025795 *
## x2           1.20929    0.36221   3.339 0.000842 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(0.5173) family taken to be 1)
##
##     Null deviance: 100.25  on 99  degrees of freedom
## Residual deviance:  84.17  on 97  degrees of freedom
## AIC: 271.38
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  0.517
##           Std. Err.:  0.150
##
##  2 x log-likelihood:  -263.376
```

We note that the results are similar.

We followed all assumptions of Negative Binomial Regression in regressing y on x1 and x2 using the test_neg_binom_regression_data data set. We will compare the residual of this regression to that of all the others where assumptions will be broken.

The residual for where all assumptions are met:

```
prediction_all_assumptions_met <-
  as.numeric(
    predict_neg_binom(
      test_neg_binom_regression_data,
      test_neg_binom_regression_data$x1,
      test_neg_binom_regression_data$x2,
      y = test_neg_binom_regression_data$y,
      implementation_neg_binom = our_implementation_neg_binom
    )
  )
residual_all_assumptions_met <- sqrt(mean((
  test_neg_binom_regression_data$y - prediction_all_assumptions_met
)^2))
residual_all_assumptions_met # small residual
```

```
## [1] 1.999863
```

```
# residual plot
plot(
  test_neg_binom_regression_data$y - prediction_all_assumptions_met,
  ylim = c(-30, 30),
  ylab = "Residuals",
  main = "Residual Plot: All assumptions met",
  pch = 16
)
abline(
  h = 0,
  col = "red",
  lty = 2,
  lwd = 3
```

```
)
```

Breaking Assumptions

Breaking the assumption that the relationship between the predictors and the log of the outcome's mean is linear

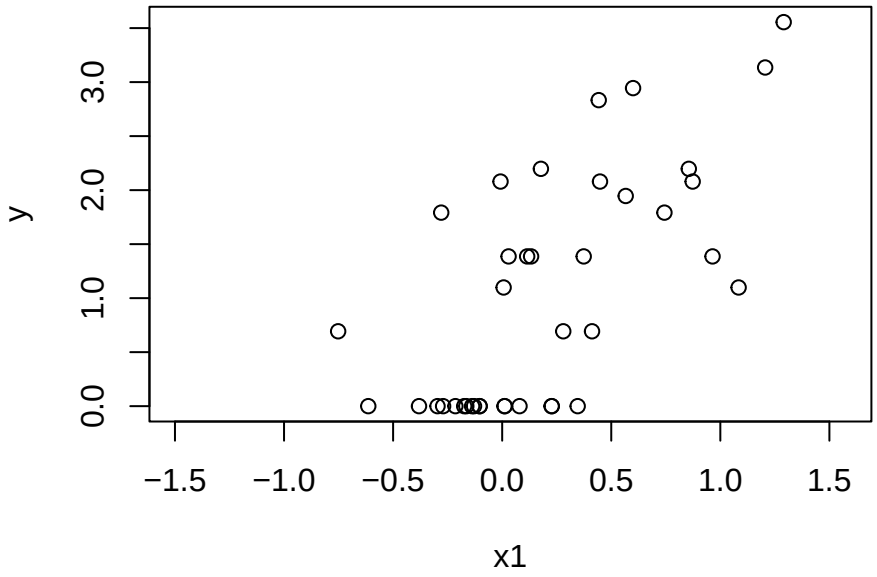
Creating a data set where, if we apply Negative Binomial regression, this assumption will be broken.

```
x1 <- rnorm(100, mean = 0, sd = 0.5)
x2 <- rnorm(100, mean = 0, sd = 0.5)
y <- rnbinom(100, mu = exp(x1 + x2)^2, size = 0.5)
test_neg_binom_regression_data_not_linear <- data.frame(x1, x2, y)
# to ensure that the variance of the outcome variable is greater
# than its mean
var(y) > mean(y)
```

```
## [1] TRUE
```

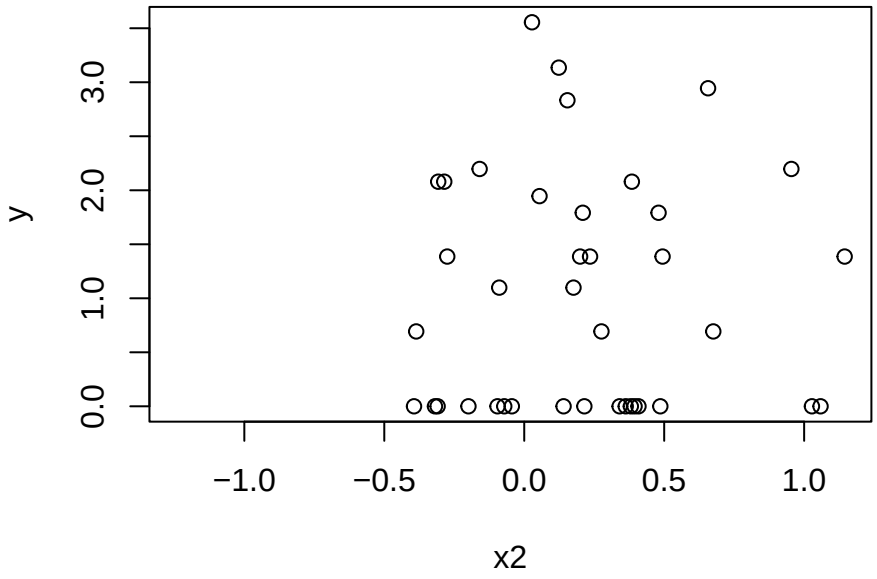
```
plot(test_neg_binom_regression_data_not_linear$x1, log(test_neg_binom_regression_data_not_linear$y),
  main = "The relationship between the log of the outcome and x1 is not linear", cex.main = 0.8,
  xlab = "x1", ylab = "y"
)
```

The relationship between the log of the outcome and x1 is not linear




```
plot(test_neg_binom_regression_data_not_linear$x2, log(test_neg_binom_regression_data_not_linear$y),
  xlab = "x2", ylab = "y", cex.main = 0.8,
  main = "The relationship between the log of the outcome and x2 is not linear"
)
```

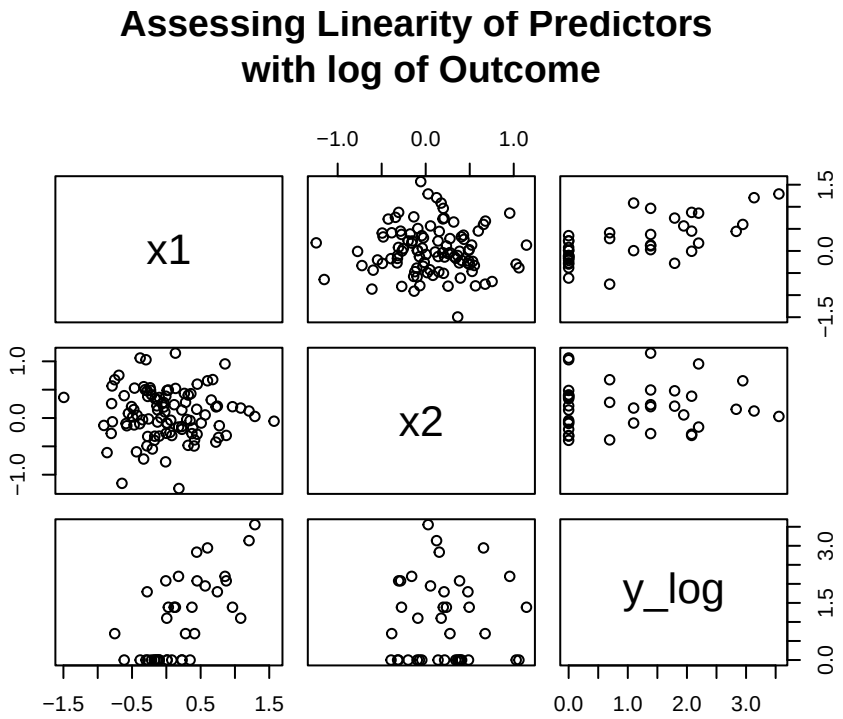
The relationship between the log of the outcome and x2 is not linear



Using our implementation of Negative Binomial to fit the model and get a residual measure.

```
our_implementation_neg_binom_not_linear <-
  negative_binomial_regression(
    test_neg_binom_regression_data_not_linear,
    test_neg_binom_regression_data_not_linear$x1,
    test_neg_binom_regression_data_not_linear$x2,
```

```
    y = test_neg_binom_regression_data_not_linear$y
  )[[2]]
```

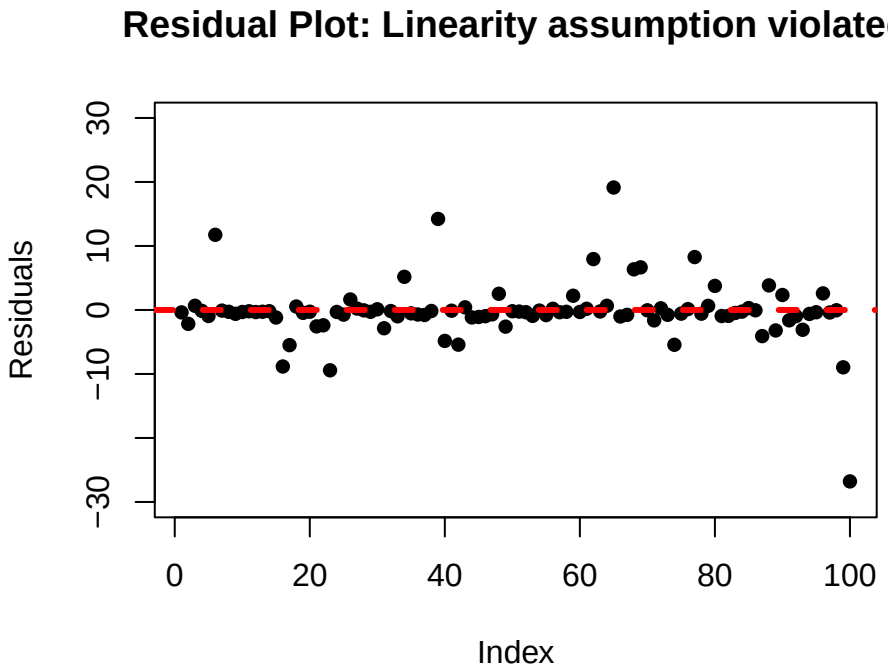



```
prediction_not_linear <-
  as.numeric(
    predict_neg_binom(
      test_neg_binom_regression_data_not_linear,
      test_neg_binom_regression_data_not_linear$x1,
      test_neg_binom_regression_data_not_linear$x2,
      y = test_neg_binom_regression_data_not_linear$y,
      implementation_neg_binom = our_implementation_neg_binom_not_linear
    )
  )
residual_not_linear <- sqrt(mean((
  test_neg_binom_regression_data_not_linear$y - prediction_not_linear
)^2))
residual_not_linear # large residual
```

```
## [1] 4.665716
```

```
# residual plot
plot(
  test_neg_binom_regression_data_not_linear$y - prediction_not_linear,
  ylim = c(-30, 30),
  ylab = "Residuals",
  main = "Residual Plot: Linearity assumption violated",
  pch = 16
)
abline(
  h = 0,
  col = "red",
  lty = 2,
```

```
  lwd = 3
)
```



Breaking the assumption that the mean of the outcome is smaller than its variance

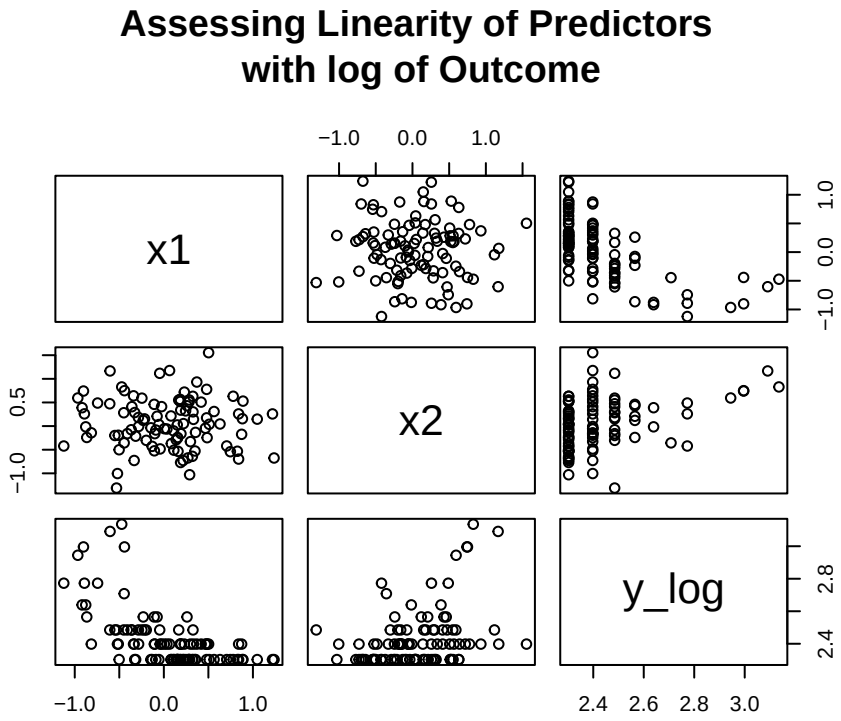
Creating a data set where, if we apply Negative Binomial regression, this assumption will be broken.

```
x1 <- rnorm(100, mean = 0, sd = 0.5)
x2 <- rnorm(100, mean = 0, sd = 0.5)
y <- rnbinom(100, mu = exp(x2 - 2 * x1), size = 100) + 10
test_neg_binom_regression_data_mean_greater <- data.frame(x1, x2, y)
# to ensure that the variance of the outcome variable is smaller
# than its mean
var(y) > mean(y)
```

```
## [1] FALSE
```

Using our implementation of Negative Binomial to fit the model and get a residual measure.

```
our_implementation_neg_binom_mean_greater <-
  negative_binomial_regression(
    test_neg_binom_regression_data_mean_greater,
    test_neg_binom_regression_data_mean_greater$x1,
    test_neg_binom_regression_data_mean_greater$x2,
    y = test_neg_binom_regression_data_mean_greater$y
  )[[2]]
```

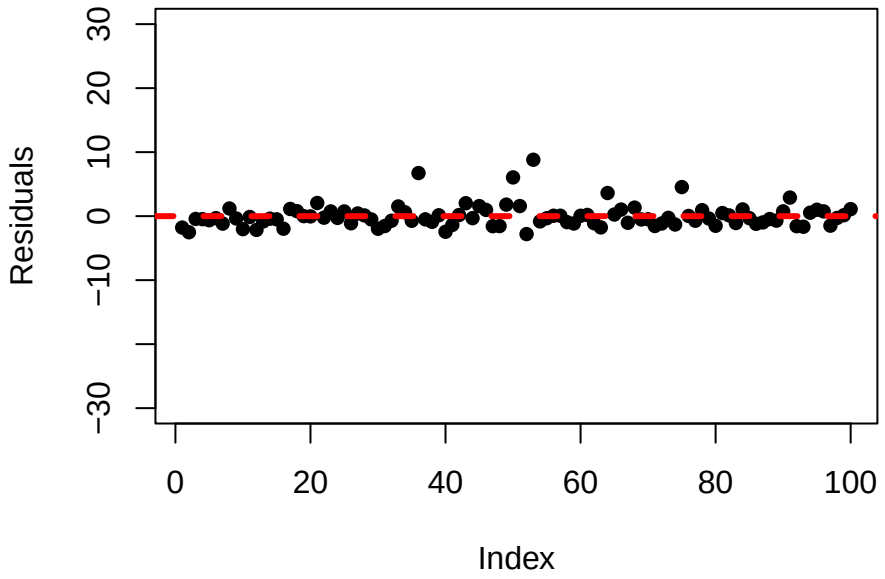



```
prediction_mean_greater <-
  as.numeric(
    predict_neg_binom(
      test_neg_binom_regression_data_mean_greater,
      test_neg_binom_regression_data_mean_greater$x1,
      test_neg_binom_regression_data_mean_greater$x2,
      y = test_neg_binom_regression_data_mean_greater$y,
      implementation_neg_binom = our_implementation_neg_binom_mean_greater
    )
  )
residual_mean_greater <- sqrt(mean((
  test_neg_binom_regression_data_mean_greater$y - prediction_mean_greater
)^2))
residual_mean_greater
```

```
## [1] 1.790385
```

```
# residual plot
plot(
  test_neg_binom_regression_data_mean_greater$y - prediction_mean_greater,
  ylim = c(-30, 30),
  ylab = "Residuals",
  cex.main = 0.9,
  main = "Residual Plot: Variance of outcome greater than mean assumption violated",
  pch = 16
)
abline(
  h = 0,
  col = "red",
  lty = 2,
  lwd = 3
)
```

Residual Plot: Variance of outcome greater than mean assumption vi



Comparing residuals when all assumptions were met versus not

```
residual_comparison <-
  t(
    data.frame(
      residual_all_assumptions_met,
      residual_not_linear,
      residual_mean_greater
    )
  )
row.names(residual_comparison) <- c(
  "All assumptions met",
  "Linearity assumption violated",
  "Variance > Mean assumption violated"
)
colnames(residual_comparison) <- "Residuals"
residual_comparison
```

```
##                                    Residuals
## All assumptions met                 1.999863
## Linearity assumption violated       4.665716
## Variance > Mean assumption violated 1.790385
```

Conclusion

The implementation of Negative Binomial Regression where all assumptions are met performs well; however, even the model where an assumption is broken; i.e. where the mean of the outcome is greater than its variance, performs well too - however, it should be noted that even though its predictions might be accurate, its standard errors and p-values might be biased.

```
set.seed(123)
library(alr4)
library(tidyverse)
```

```r
library(MASS)
library(pscl)
library(glmbb) # for crabs data
library(kableExtra)
library(lmtest)
```

Logistic Regression

Introduction

Logistic regression is used when the outcome variable is discrete and binary, which is called classification. Multinomial logistic regression can classify observations into more than two categories, but we are only doing simple logistic regression here, with two categories. We use the inverse logit function to model the probability that $Y_i = 1$.

$$logit^{-1}(x) = \frac{e^x}{1+e^x} Pr(y_i = 1) = logit^{-1}(X_i\beta)$$

`plogis` is the invlogit function.

Uses

Logistic regression is good for when covariates are continuous, as the outcome variables are bounded between 0 and 1, through the logit link.

Assumptions

1. For binary logistic regression, that outcome variables are binary
2. Independence of errors
3. Linear relationship between the outcome variable and log odds of the predictor variables
4. No multicollinearity

Our Logistic Regression Implementation

```r
library(alr4)
# invlogit <- plogis

logistic_function <- function(fn_formula, data, predict = F) {
  number_omitted <- nrow(data) - nrow(na.omit(data))
  data <- na.omit(data)

  vars <- all.vars(as.formula(fn_formula))
  y_name <- vars[1]
  x_name <- vars[2:length(vars)]
  n <- nrow(data)
  Y <- matrix(data[, y_name], nrow = n, ncol = 1)
  X <- matrix(cbind(rep(1, n)))

  # take in categorical data
  var_names <- vector("character")
  for (i in x_name) {
    if (suppressWarnings(all(!is.na(as.numeric(as.character(data[, i])))))) {
      X <- cbind(X, as.numeric(as.character(data[, i])))
```

```
      var_names <- c(var_names, i)
    } else {
      categories <- sort(unique(data[, i]))
      for (j in categories[2:length(categories)]) {
        new_col_name <- paste0(i, j)
        new_col <- ifelse(data[, i] == j, 1, 0)
        X <- cbind(X, new_col)
        var_names <- c(var_names, new_col_name)
      }
    }
  }
  optim_logistic <- function(beta, X, Y) {
    beta <- as.matrix(beta, nrow = 4)
    pi <- plogis(X %*% beta)
    loglikelihood <- -sum(Y * log(pi) + (1 - Y) * log(1 - pi))
    return(loglikelihood)
  }
  result <- optim(par = rep(0, ncol(X)), fn = optim_logistic, X = X, Y = Y, hessian = T)
  OI <- solve(result$hessian)
  se <- sqrt(diag(OI))
  t_statistic <- result$par / se
  df <- nrow(X) - ncol(X)
  p_value <- 2 * pnorm(-1 * abs(t_statistic))
  # https://stats.stackexchange.com/questions/52475/how-are-the-p-values-of-the-glm-in-r-calculated

  coef <- rbind(result$par, se, t_statistic, p_value)
  colnames(coef) <- c("(Intercept)", var_names)
  rownames(coef) <- c("Estimate", "Std. Error", "z value", "p value")
  coef <- t(coef)

  b_hat <- result$par
  predictions <- plogis(X %*% b_hat)

  if (predict) {
    return(predictions)
  } else {
    return(coef)
  }
}
```

Creating testing data set with a single predictor variable to compare our implementation with `glm` logistic function

```
# create fake data
x1 <- rnorm(100, 2, 1)
prob <- plogis(-1 + 0.5 * x1)
y <- rbinom(100, 1, prob)
sim_data <- data.frame(y, x1)

# compare DIY logistic function with glm
fit_sim_data_1 <- glm(y ~ x1, data = sim_data, family = binomial)
summary(fit_sim_data_1)$coef
```

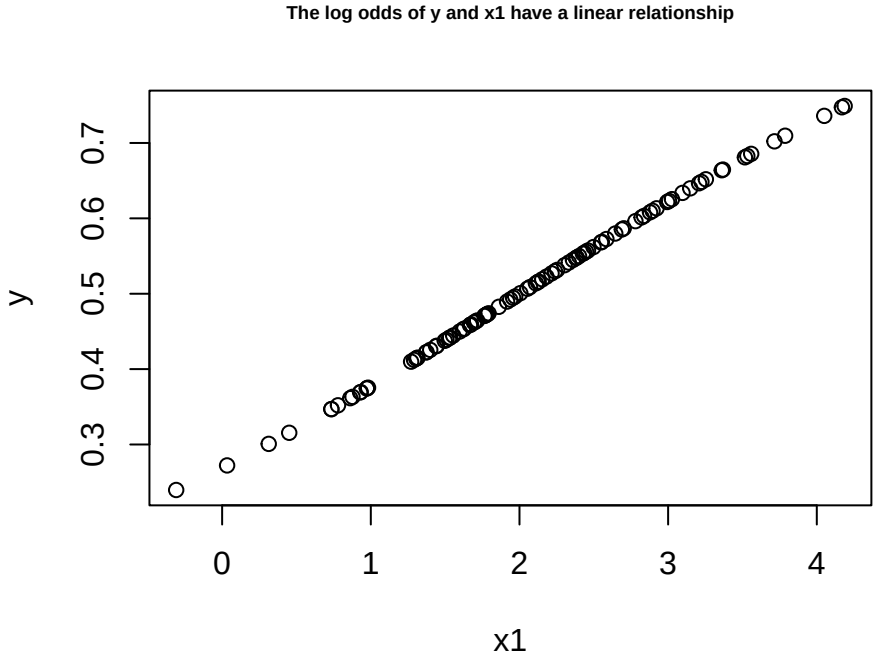
```
##               Estimate Std. Error   z value   Pr(>|z|)
## (Intercept) -0.7456721  0.5316508 -1.402560 0.16074811
```

```
## x1           0.5410003  0.2427422  2.228703 0.02583366
```

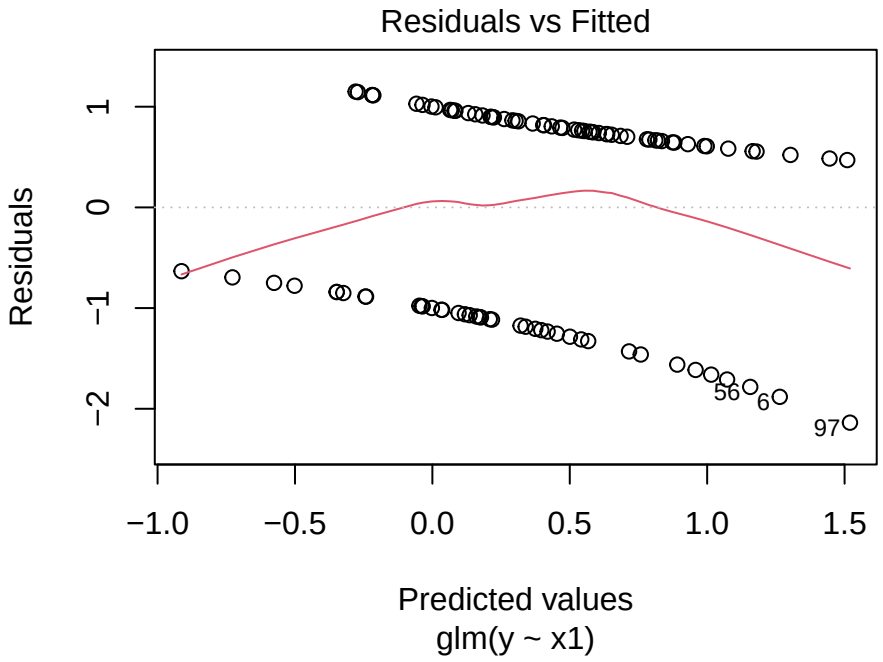
```
logistic_function(fn_formula = "y ~ x1", data = sim_data)
```

```
##                Estimate Std. Error   z value    p value
## (Intercept) -0.7459709  0.5316604 -1.403097 0.16058801
## x1           0.5410940  0.2427464  2.229051 0.02581053
```

```
# checking for linear relationship
plot(sim_data$x1, prob,
  main = "The log odds of y and x1 have a linear relationship", cex.main = 0.6,
  xlab = "x1", ylab = "y"
)
```




```
# check for correlation of residuals vs fit
plot(fit_sim_data_1, which = 1)
```



Creating testing data set with multiple covariates to compare our implementation with `glm` logistic function

```
# create fake data with multiple x's
x1 <- rnorm(100, 2, 1)
x2 <- rnorm(100, 4, 1)
x3 <- rnorm(100, 6, 1)
prob <- plogis(-1 + x1 + x2 - 0.5 * x3)
y <- rbinom(100, 1, prob)
sim_data <- data.frame(y, x1, x2, x3)

# compare DIY logistic function with glm
fit_sim_data <- glm(y ~ x1 + x2 + x3, data = sim_data, family = binomial)
summary(fit_sim_data)$coef
```

```
##               Estimate Std. Error    z value    Pr(>|z|)
## (Intercept)  2.4011218  2.4617714  0.9753634 0.329380012
## x1           1.3097873  0.4646174  2.8190663 0.004816356
## x2           0.9528707  0.3591008  2.6534911 0.007966387
## x3          -1.1133007  0.3837826 -2.9008631 0.003721364
```

```
logistic_function(fn_formula = "y ~ x1 + x2 + x3", data = sim_data)
```

```
##               Estimate Std. Error    z value     p value
## (Intercept)  2.4003220  2.4618335  0.9750139 0.329553343
## x1           1.3102691  0.4646817  2.8197132 0.004806659
## x2           0.9530523  0.3591195  2.6538587 0.007957715
## x3          -1.1134496  0.3838115 -2.9010321 0.003719358
```

Using the alr4 Donner data to test categorical data, and compare our implementation with `glm` logistic function

```
Donner$survived <- Donner$y == "survived"
fit_Donner <- glm(survived ~ age + sex + status, data = Donner, family = "binomial")
summary(fit_Donner)$coef
```

```
##                 Estimate   Std. Error      z value    Pr(>|z|)
```

```
## (Intercept)    1.4873491 4.926432e-01  3.019120666 0.002535095
## age           -0.0281043 1.504262e-02 -1.868311545 0.061718658
## sexMale       -0.7279889 5.172218e-01 -1.407498622 0.159279585
## statusHired   -0.5985928 6.281956e-01 -0.952876467 0.340652665
## statusSingle -17.4555740 1.765537e+03 -0.009886838 0.992111573
```

```
logistic_function(fn_formula = "survived ~ age + sex + status", data = Donner)
```

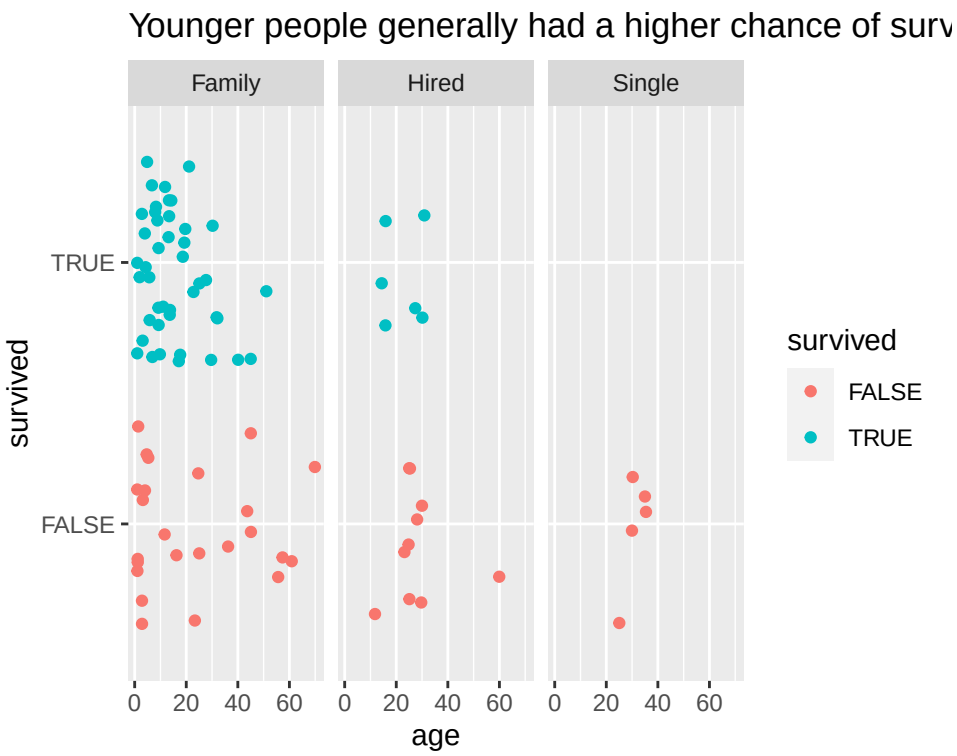
```
##                  Estimate  Std. Error    z value     p value
## (Intercept)   1.50395496  0.49431184  3.0425226 0.002346042
## age          -0.02838637  0.01507534 -1.8829672 0.059704810
## sexMale      -0.74855738  0.51805480 -1.4449386 0.148475138
## statusHired  -0.58665853  0.62833280 -0.9336748 0.350471646
## statusSingle -6.48117449 12.14886686 -0.5334798 0.593701523
```

Interpretation of the coefficients

A 1-unit difference in age corresponds to -0.02 in the logit probability of having survived in the Donner party, or a multiplicative change of $e^{-0.0283} = 0.972$ in the odds of surviving.

```
ggplot(Donner) +
  geom_jitter(aes(x = age, y = survived, color = survived)) +
  facet_wrap(vars(status)) +
  ggtitle("Younger people generally had a higher chance of surviving")
```

```
## Warning: Removed 3 rows containing missing values (`geom_point()`).
```



Show that our implementation of logistic regression can also make predictions

```
idx <- sample(1:nrow(Donner), 5)
p1 <- logistic_function(fn_formula = "survived ~ age + sex + status", data = Donner, predict = T)[idx, ]
p2 <- predict(fit_Donner, type = "response")[idx]
compare_predict_data <- data.frame(p1, p2)
```

```r
colnames(compare_predict_data) <- c("Our implementation", "GLM")

kable(compare_predict_data, digits = 3, caption = "Comparison of logistic prediction", booktabs = TRUE,
```

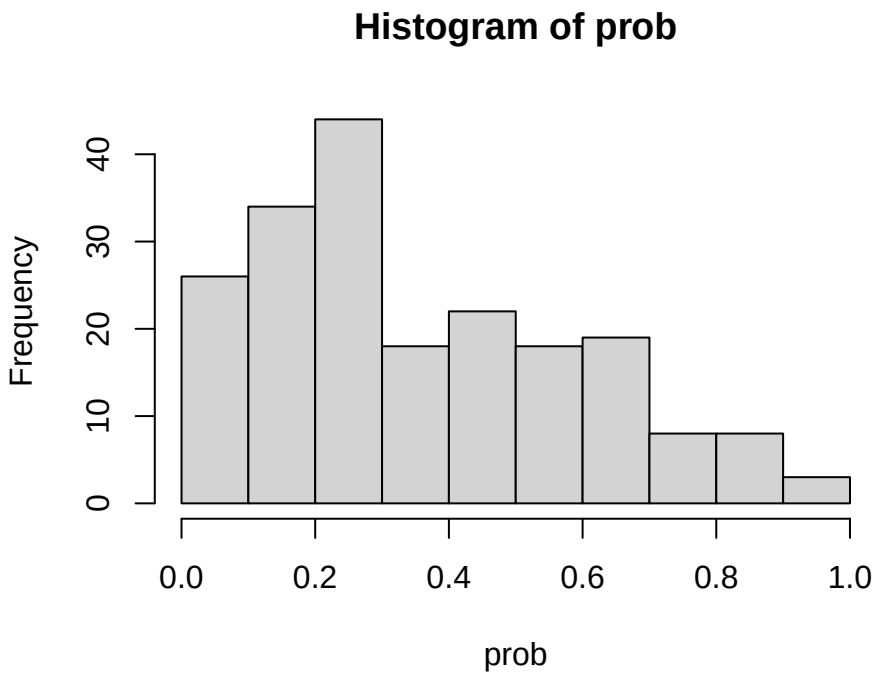
Table 1: Comparison of logistic prediction

	Our implementation	GLM
Breen_Margaret_Isabella	0.814	0.811
Breen_Patrick	0.334	0.338
Spitzer_Augustus	0.001	0.000
Donner_Isaac	0.649	0.650
Foster_Sarah_Ann_Charlotte_Murphy	0.724	0.722

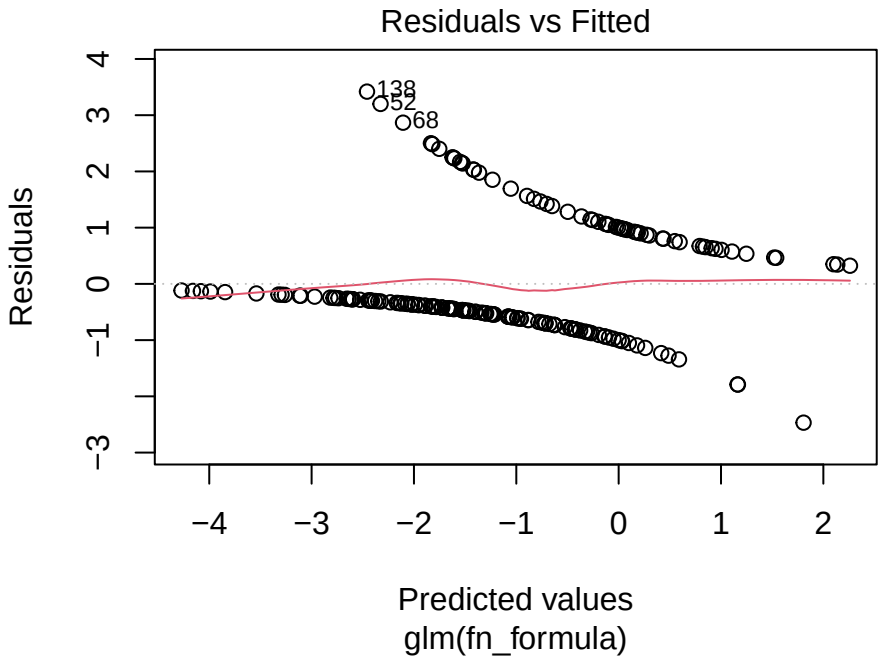
Function to check assumptions

```r
test_logistic_assumptions <- function(fn_formula, data) {
  n <- nrow(data)
  vars <- all.vars(as.formula(fn_formula))
  y_name <- vars[1]
  Y <- data[, y_name]

  # outcome variables are binary
  if (length(unique(Y)) == 2) {
    assp_1 <- paste("Binary outcomes assumption is met.")
  } else {
    return(paste("Binary outcomes assumption is not satisfied. There are", length(unique(Y)), "outcomes
  }

  x_name <- vars[2:length(vars)]
  X <- data[, x_name]
  preds <- logistic_function(fn_formula = fn_formula, data = data, predict = T) # predictions are in pr
  logit_vals <- log(preds / (1 - preds))
  plot_data <- data.frame(logit_vals, X) |> gather(key = "predictors", value = "predictor_value", -logi

  # Linear relationship between the outcome variable and log odds of the predictor variables
  assp_2 <- ggplot(plot_data, aes(logit_vals, predictor_value)) +
    geom_point(size = 0.5, alpha = 0.5) +
    geom_smooth(method = "loess") +
    theme_bw() +
    facet_wrap(~predictors, scales = "free_y")

  # Independence of errors
  # plot residuals vs fits
  model <- glm(fn_formula, data = data, family = binomial)
  assp_3 <- plot(model, which = 1)

  # No multicollinearity
  assp_4 <- cor(data[, -1])

  predicted.values <- ifelse(preds >= 0.5, 1, 0)
  check_perf <- data.frame(Y, predicted.values) |> mutate(correct = Y == predicted.values)
```

```
  check_perf <- paste0(mean(check_perf$correct) * 100, "% classified correctly")
  return(list(assp_1, assp_2, print(assp_3), assp_4, check_perf))
}

n <- 200
x1 <- rnorm(n, 2, 1)
x2 <- rnorm(n, 4, 1)
prob <- plogis(-1 + 1.2 * x1 - 0.5 * x2)
hist(prob)
```



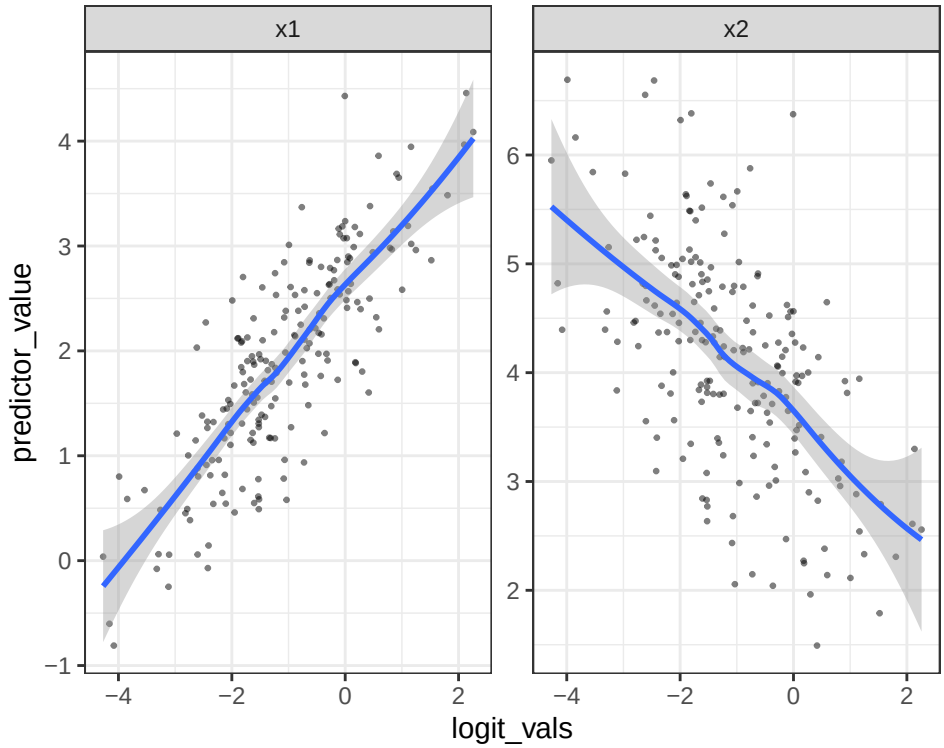

```
y <- rbinom(n, 1, prob)
sim_data <- data.frame(y, x1, x2)

test_logistic_assumptions(fn_formula = "y ~ x1 + x2", data = sim_data)
```




```
## NULL
```

```
## [[1]]
## [1] "Binary outcomes assumption is met."
##
## [[2]]
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




```
##
```

```
## [[3]]
## NULL
##
## [[4]]
##           x1        x2
## x1  1.000000 -0.018217
## x2 -0.018217  1.000000
##
## [[5]]
## [1] "78.5% classified correctly"
```

1. For binary logistic regression, that outcome variables are binary

   Check how many unique outcome variables there are.

2. Independence of errors

   Check residual plots.

3. Linear relationship between the outcome variable and log odds of the predictor variables

   Check scatterplots of log odds vs predictor variables to see that there is an approximately linear relationship.

4. No multicollinearity

   Look at the correlation matrix, generally any value over 0.9 is problematic.

Breaking assumptions

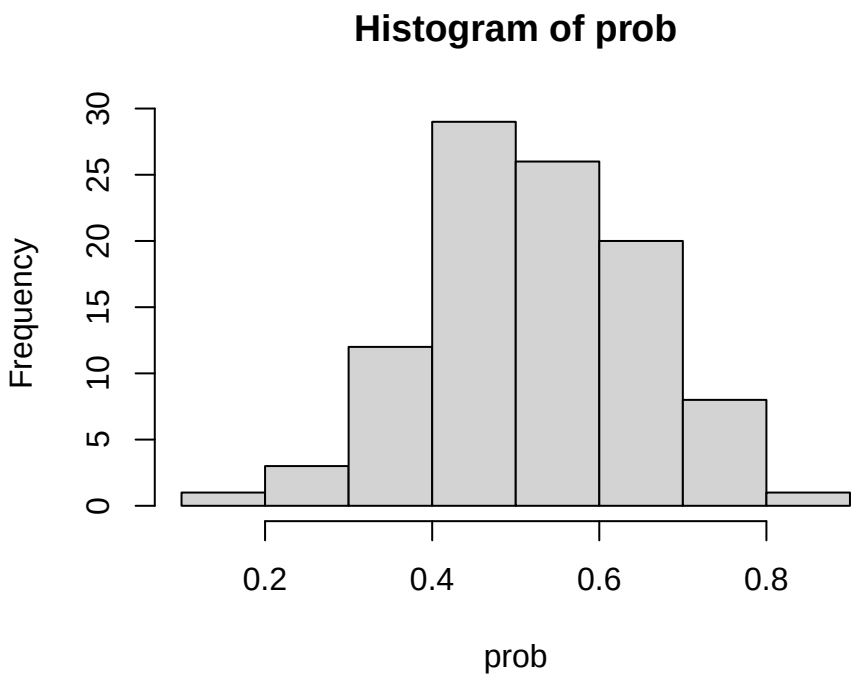
```
n <- 100
x1 <- rnorm(n, 2, 1)
prob <- plogis(-1 + 0.8 * x1)
y <- rpois(n, prob)
bad_data <- data.frame(y, x1)

test_logistic_assumptions(fn_formula = "y ~ x1", data = bad_data)
```

1. For binary logistic regression, that outcome variables are binary

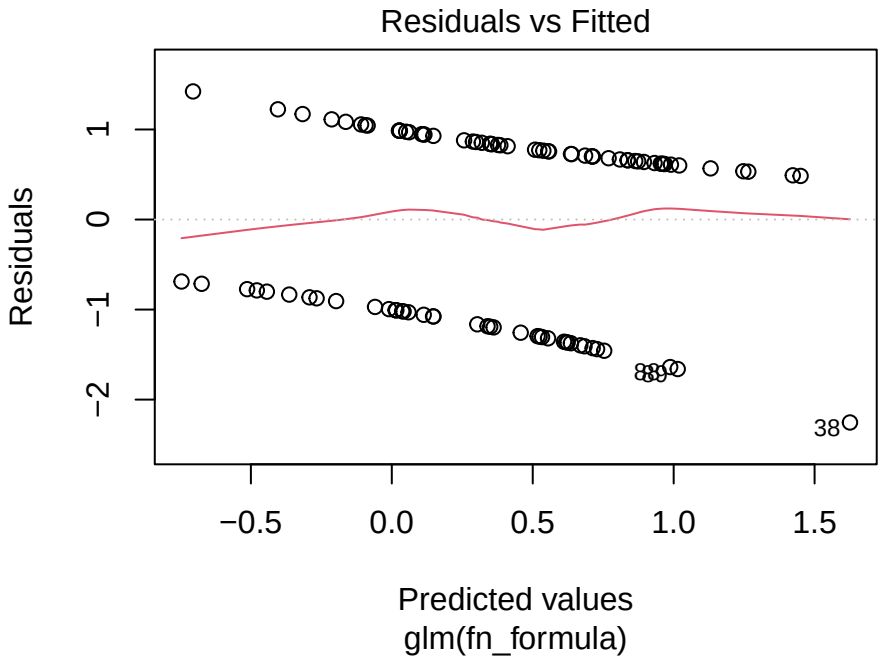
```
## [1] "Binary outcomes assumption is not satisfied. There are 4 outcomes."
```

```
n <- 100
x1 <- rnorm(n, 2, 1)
x2 <- rnorm(n, 0, 1)
prob <- plogis(-1.2 + 0.4 * x1 + 0.3 * x2 + rbeta(n, 2, 2))
hist(prob)
```

2. Independence of errors



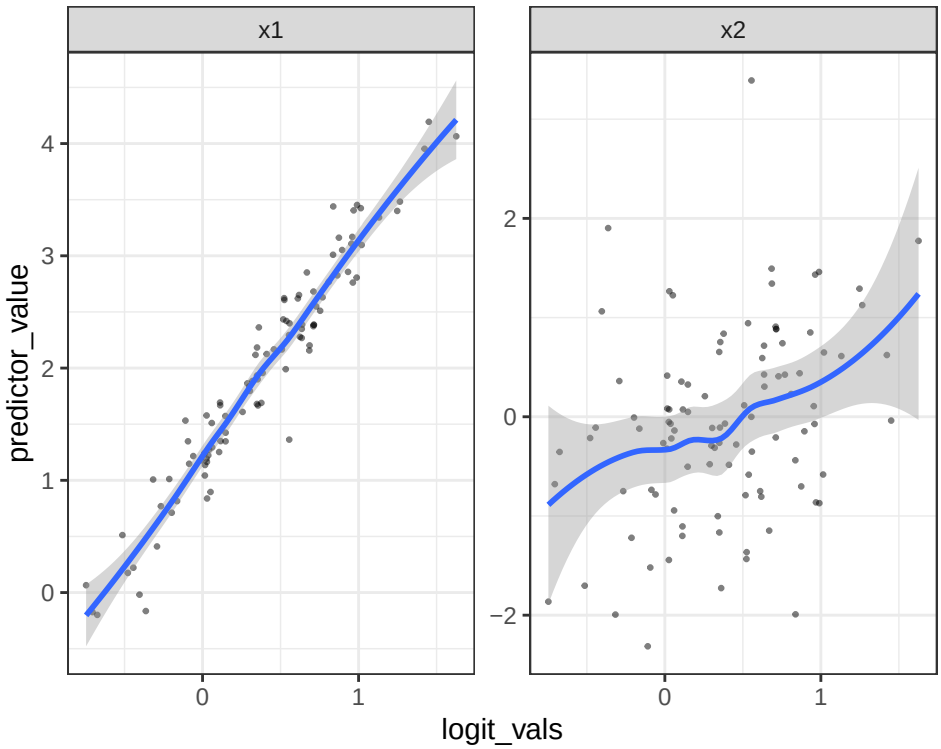

```
y <- rbinom(n, 1, prob)
bad_data <- data.frame(y, x1, x2)

test_logistic_assumptions(fn_formula = "y ~ x1 + x2", data = bad_data)
```




```
## NULL
```

```
## [[1]]
## [1] "Binary outcomes assumption is met."
##
## [[2]]
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

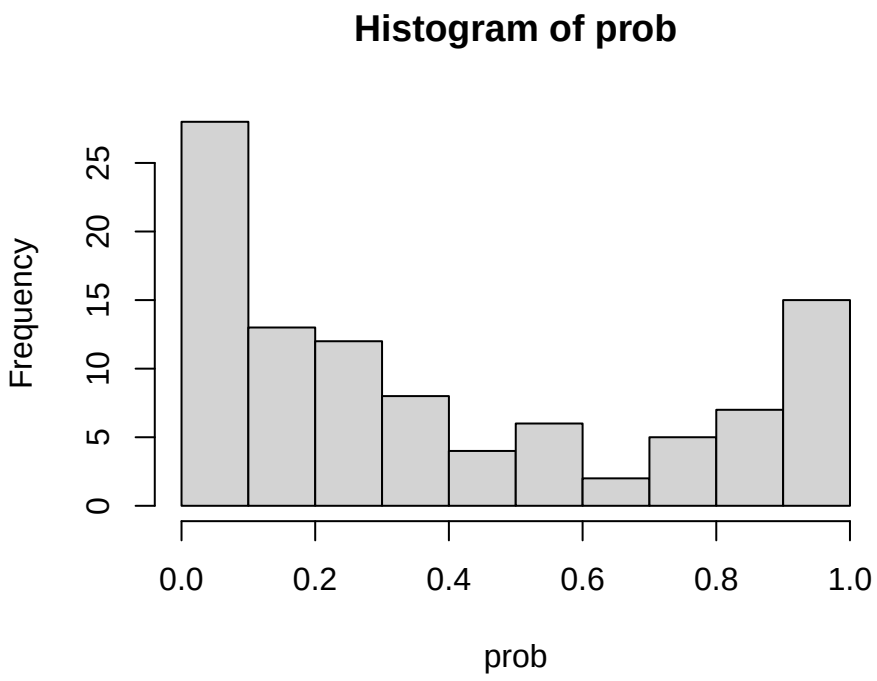



```
##
## [[3]]
## NULL
##
## [[4]]
##             x1         x2
## x1 1.00000000 0.09361225
## x2 0.09361225 1.00000000
##
## [[5]]
## [1] "62% classified correctly"
```

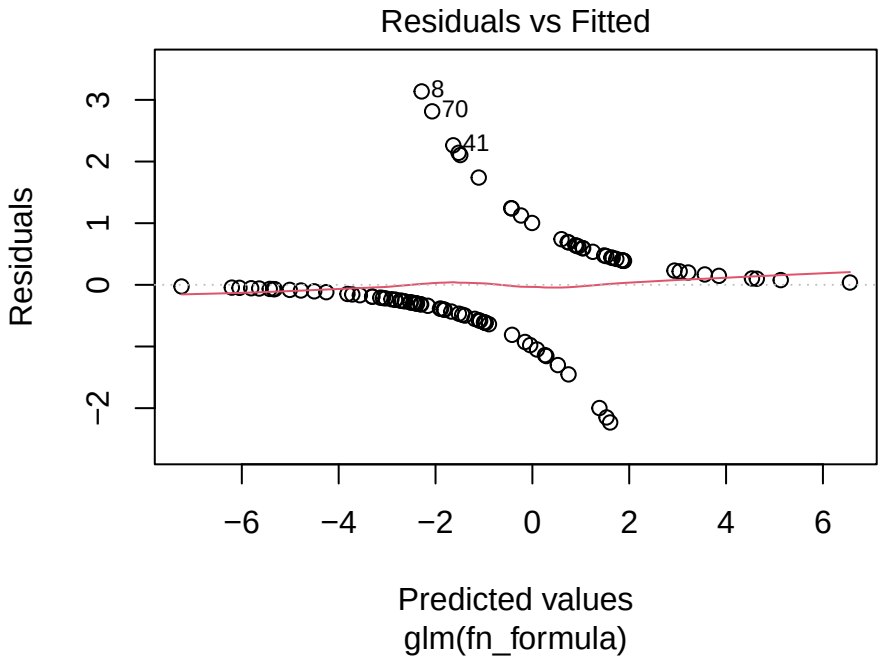
I used `rbeta` to introduce more error for the middle of the probabilities, which shows in the residual plot, where it looks like there is a peak in the fitted line around 0.

```
n <- 100
x1 <- rnorm(n, 0, 1)
x2 <- rnorm(n, 2, 1)
x3 <- rnorm(n, -2, 1)
prob <- plogis(-1 + x1 * x2 + 1.3 * x2 - 0.5 * x3^2)
hist(prob)
```

3. Linear relationship between the outcome variable and log odds of the predictor variables



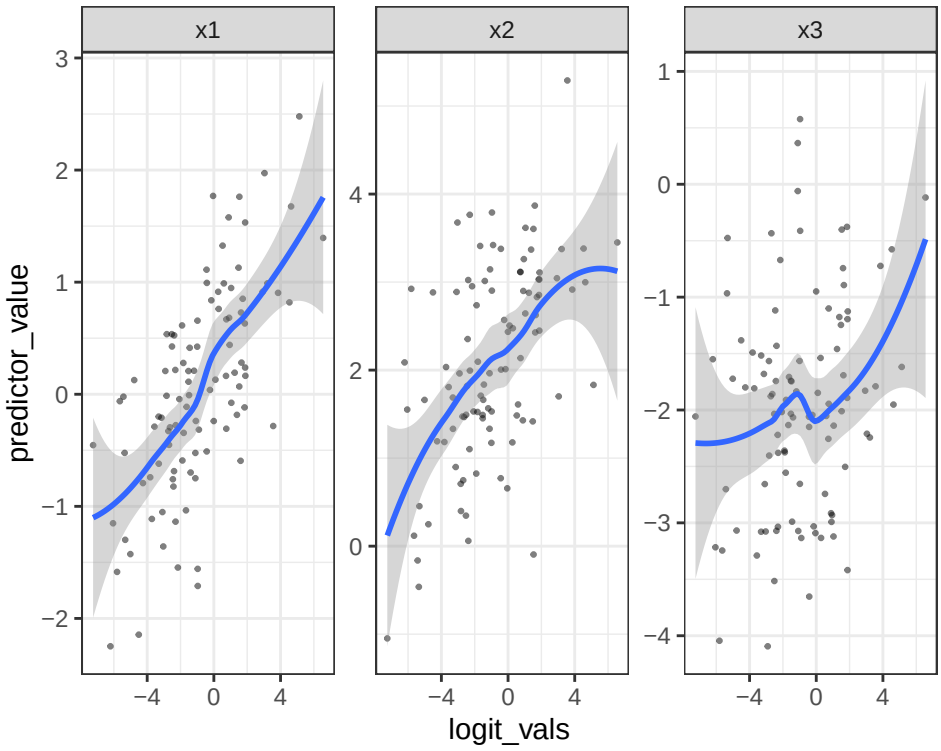

```
y <- rbinom(n, 1, prob)
bad_data <- data.frame(y, x1, x2, x3)

test_logistic_assumptions(fn_formula = "y ~ x1 + x2 + x3", data = bad_data)
```




```
## NULL
```

```
## [[1]]
## [1] "Binary outcomes assumption is met."
##
## [[2]]
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

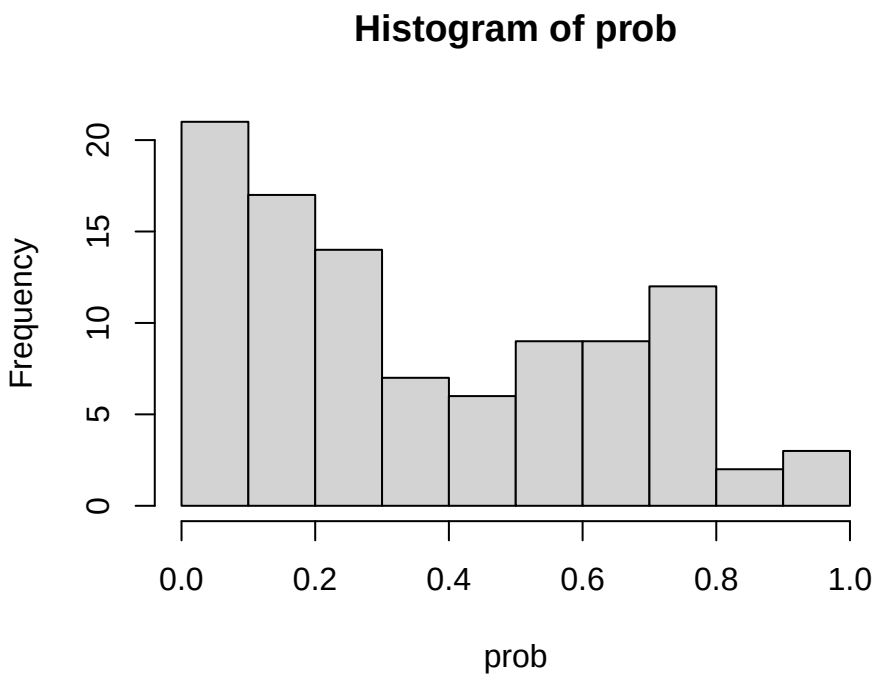



```
##
## [[3]]
## NULL
##
## [[4]]
##             x1          x2          x3
## x1  1.0000000 -0.16860022 -0.17308515
## x2 -0.1686002  1.00000000  0.02992617
## x3 -0.1730851  0.02992617  1.00000000
##
## [[5]]
## [1] "82% classified correctly"
```

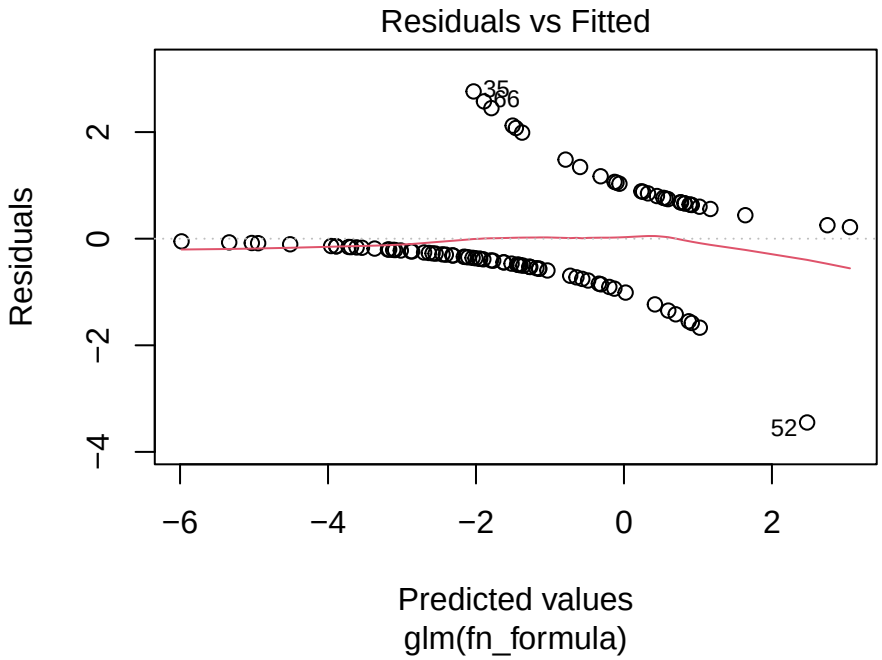
For the probability used in the DGP, it is no longer a linear relationship of $X_i\beta$ but one that involves interactions and squared variables. The non-linear relationship is visible in the plots of the outcome variable vs log odds of each predictor variable.

```
n <- 100
x1 <- rnorm(n, 0, 1)
x2 <- rnorm(n, x1, 0.2)
prob <- plogis(-1 + x1 + 0.5 * x2)
hist(prob)
```

4. No multicollinearity



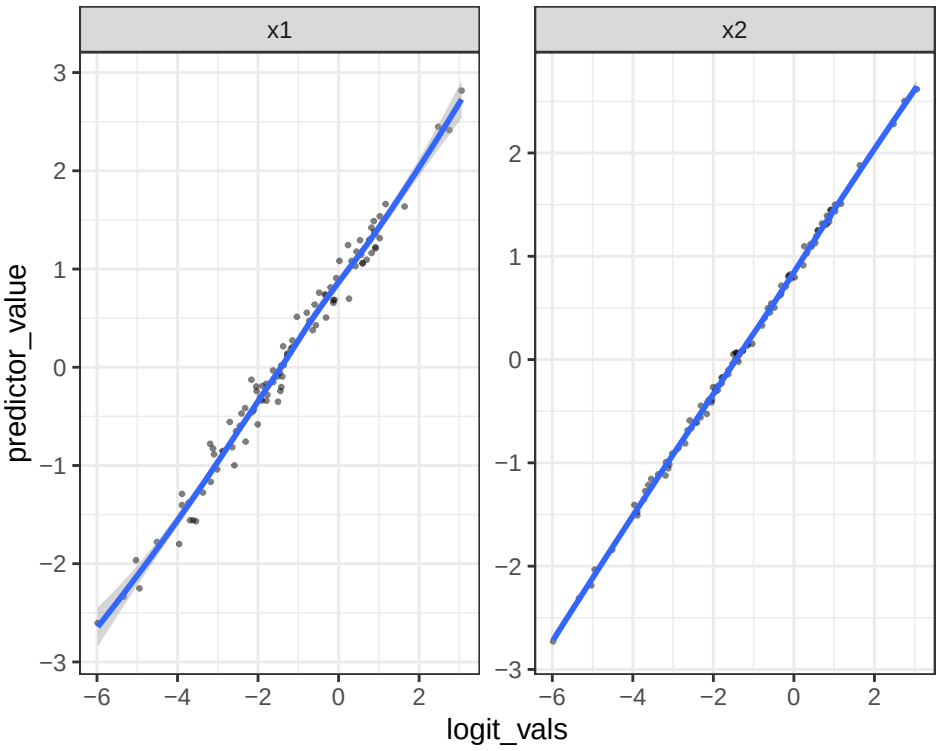

```
y <- rbinom(n, 1, prob)
bad_data <- data.frame(y, x1, x2)

test_logistic_assumptions(fn_formula = "y ~ x1 + x2", data = bad_data)
```




```
## NULL
```

```
## [[1]]
## [1] "Binary outcomes assumption is met."
##
## [[2]]
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




```
##
## [[3]]
## NULL
##
## [[4]]
##           x1        x2
## x1 1.0000000 0.9847734
## x2 0.9847734 1.0000000
##
## [[5]]
## [1] "80% classified correctly"
```

The DGP involves $x2$ being created from an `rnorm` around the mean of $x1$, and a small SD so that it is even more correlated. The correlation matrix shows a problematically high correlation. The model still classifies fairly well, but what happens with multicollinearity is that it becomes difficult for the model to estimate the relationship between each predictor variable and the outcome variable independently because the predictor variables tend to change in unison. The p-values are also less trustworthy.

Next Steps

- Instead of just using maximum likelihood, we could try using iteratively reweighted least squares or Newton-Ralphson.

Conclusion

Breaking certain assumptions do not make the logistic model classify worse. Independence of errors seems to be the worst case, but the rest do not change the correct classification rate much.

Poisson Regression

Introduction

Poisson regression is used for count and rate data. We use Poisson distribution to model the expected value of Y, which is denoted by $E(Y) = \mu$. The identity link is the log link, so the Poisson regression model for counts is $log(\mu) = \alpha + \beta x$. The Poisson distribution with parameter λ, $Poi(\lambda)$ has the probability mass function

$$P(X = k) = exp(-\lambda)\frac{\lambda^k}{k!}, k = 0, 1, 2, 3, ...$$

Uses

Poisson regression can be used for count data, such as number of asthmatic attacks in one year based on the number of hospital admissions and systolic blood pressure. When the predictor variables are continuous, poisson regression ensures that the outcome variable is positive, compared to a linear regression which might predict negative counts. Another use case for Poisson regression is when the number of cases is small relative to the number of no events, such as when the number of deaths due to COVID-19 are small relative to the total population size. Logistic regression is more useful when we have data on both the binary outcomes (e.g. death and non-deaths).

Assumptions

- outcome variable must be count data
- Independent observations
- Distribution of counts follow a Poisson distribution
- No overdispersion - the mean and variance of the distribution are equal. If the variance is greater than the mean, negative binomial regression may be more appropriate

Our Poisson Regression Implementation

```
poisson_function <- function(fn_formula, data, predict = F) {
  number_omitted <- nrow(data) - nrow(na.omit(data))
  data <- na.omit(data)

  vars <- all.vars(as.formula(fn_formula))
  y_name <- vars[1]
  x_name <- vars[2:length(vars)]
  n <- nrow(data)
  Y <- matrix(data[, y_name], nrow = n, ncol = 1)
  X <- matrix(cbind(rep(1, n)))

  # take in categorical data
  var_names <- vector("character")
  for (i in x_name) {
    if (suppressWarnings(all(!is.na(as.numeric(as.character(data[, i])))))) {
      X <- cbind(X, as.numeric(as.character(data[, i])))
      var_names <- c(var_names, i)
    } else {
      categories <- sort(unique(data[, i]))
      for (j in categories[2:length(categories)]) {
        new_col_name <- paste0(i, j)
        new_col <- ifelse(data[, i] == j, 1, 0)
        X <- cbind(X, new_col)
```

```
        var_names <- c(var_names, new_col_name)
      }
    }
  }
  optim_poisson <- function(beta, X, Y) {
    beta <- as.matrix(beta, nrow = 4)
    beta_x <- X %*% beta
    loglikelihood <- -sum(Y * beta_x - exp(beta_x))
    return(loglikelihood)
  }
  result <- optim(par = rep(0, ncol(X)), fn = optim_poisson, X = X, Y = Y, hessian = T)
  OI <- solve(result$hessian)
  se <- sqrt(diag(OI))
  z_value <- result$par / se
  df <- nrow(X) - ncol(X)
  p_value <- 2 * pnorm(-1 * abs(z_value))

  coef <- rbind(result$par, se, z_value, p_value)
  colnames(coef) <- c("(Intercept)", var_names)
  rownames(coef) <- c("Estimate", "Std. Error", "z value", "p value")

  b_hat <- result$par
  predictions <- exp(X %*% b_hat)

  if (predict) {
    return(predictions)
  } else {
    return(t(coef))
  }
}
```

Testing poisson implementation with simulated data

```
n <- 100
x1 <- sample(0:1, n, replace = T)
lambda <- exp(2 + 0.5 * x1)
y <- rpois(n, lambda)
sim_data <- data.frame(y, x1)
m1 <- glm(y ~ x1, family = poisson, data = sim_data)
summary(m1)$coef
```

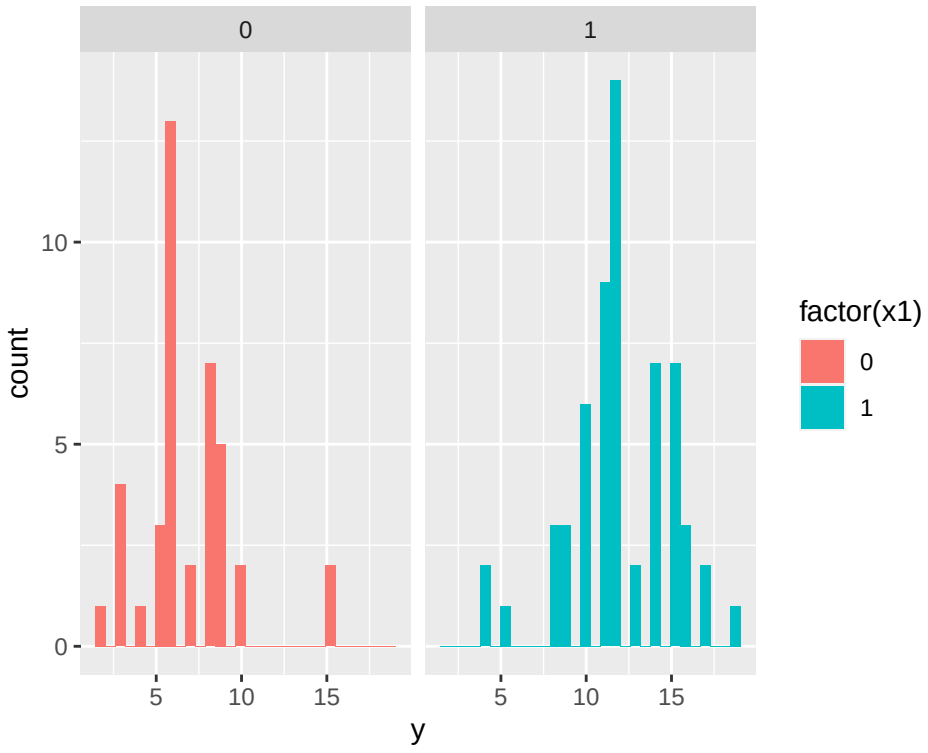
```
##              Estimate Std. Error   z value      Pr(>|z|)
## (Intercept) 1.9315214 0.06019282 32.088898 6.298177e-226
## x1          0.5547732 0.07078233  7.837735  4.587454e-15
```

```
poisson_function(fn_formula = "y ~ x1", data = sim_data)
```

```
##              Estimate Std. Error   z value       p value
## (Intercept) 1.9314943 0.06019373 32.087965 6.489899e-226
## x1          0.5547785 0.07078331  7.837702  4.588663e-15
```

```
ggplot(sim_data) +
  geom_histogram(aes(x = y, fill = factor(x1))) +
  facet_wrap(~x1)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Show that our implementation of poisson regression can also make predictions

```
idx <- sample(1:n, 5)
p1 <- poisson_function(fn_formula = "y ~ x1", data = sim_data, predict = T)[idx]
p2 <- predict(m1, type = "response")[idx]

compare_predict_data <- data.frame(p1, p2)
colnames(compare_predict_data) <- c("Our implementation", "GLM")

kable(compare_predict_data, digits = 3, caption = "Comparison of Poisson prediction", booktabs = TRUE, 
```

Table 2: Comparison of Poisson prediction

	Our implementation	GLM
100	12.016	12.017
97	6.900	6.900
34	12.016	12.017
1	12.016	12.017
11	12.016	12.017

Testing poisson implementation with `crabs` data

```
# comparing coefficients with crabs data
data(crabs, package = "glmbb")
summary(glm(satell ~ width, family = poisson(link = "log"), data = crabs))$coef
```

```
##               Estimate Std. Error   z value     Pr(>|z|)
## (Intercept) -3.3047572 0.54224155 -6.094622 1.096964e-09
## width        0.1640451 0.01996535  8.216491 2.095450e-16
```

```r
# a bit over-dispersed
mean(crabs$satell)
```

```
## [1] 2.919075
```

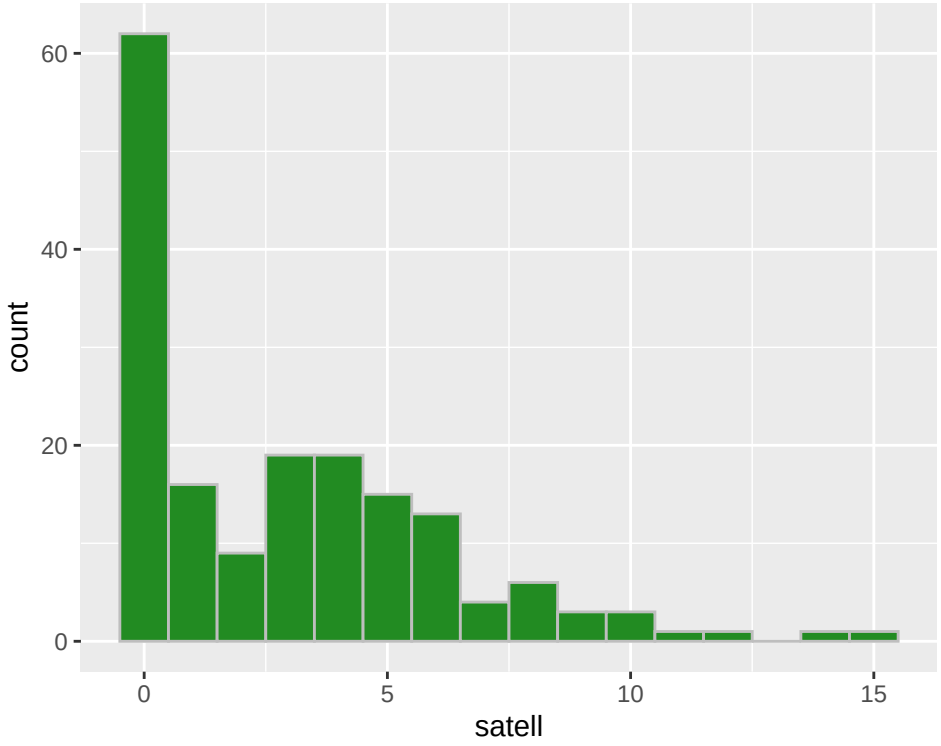
```r
var(crabs$satell)
```

```
## [1] 9.912018
```

```r
poisson_function(fn_formula = "satell ~ width", data = crabs)
```

```
##               Estimate Std. Error   z value      p value
## (Intercept) -3.3041041 0.54211127 -6.094882 1.095184e-09
## width        0.1640204 0.01995812  8.218226 2.065351e-16
```

```r
ggplot(crabs) +
  geom_histogram(aes(x = satell),
    binwidth = 1,
    fill = "forestgreen", color = "gray"
  )
```



Interpretation of coefficients

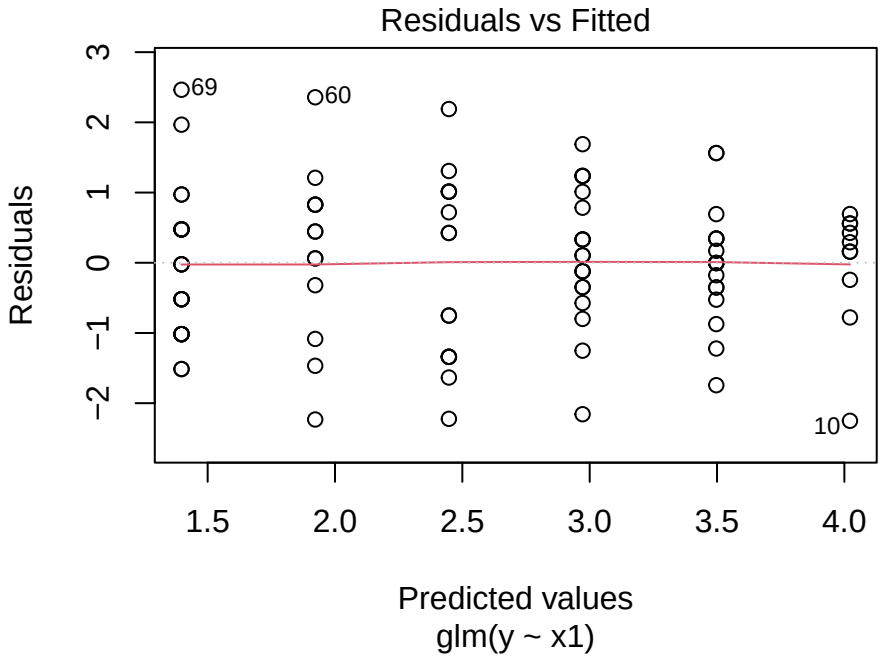
A change in 1 unit of width has a multiplicative effect on the mean of Y. For a 1 unit increase in log(width), the estimated mean number of satellites increases by a factor of $e^{0.164} = 1.178$ when the log linear model is $log(\mu_i) = -3.3 + 0.164 * width_i$.

Breaking Assumptions

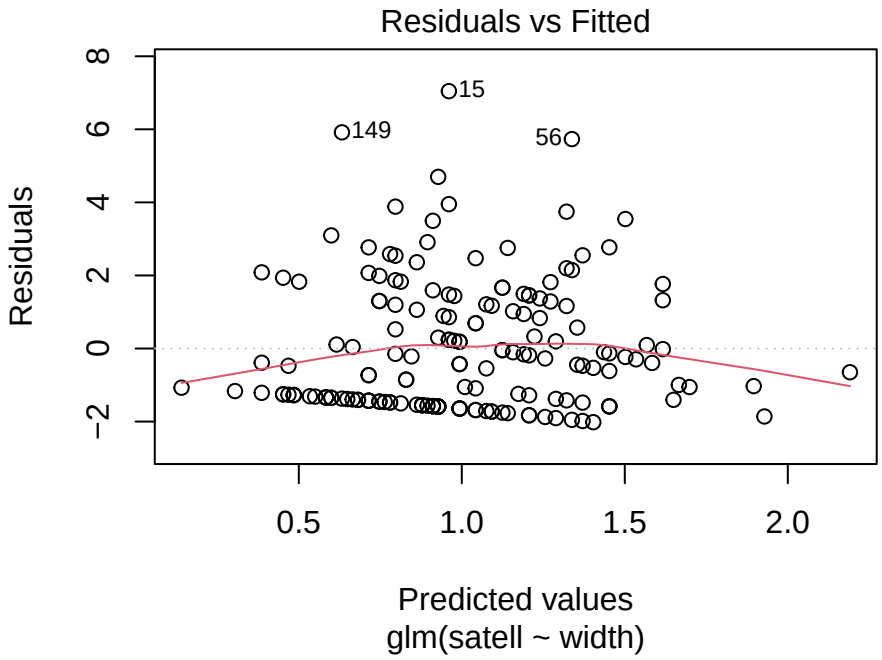
```r
n <- 100
x1 <- sample(0:5, n, replace = T)
lambda <- exp(1.5 + 0.5 * x1)
```

```
y <- rpois(n, lambda)
sim_data <- data.frame(y, x1)
mp <- glm(y ~ x1, data = sim_data, family = poisson)
plot(mp, which = 1)
```

Independent observations




```
mp <- glm(satell ~ width, data = crabs, family = poisson)
plot(mp, which = 1)
```



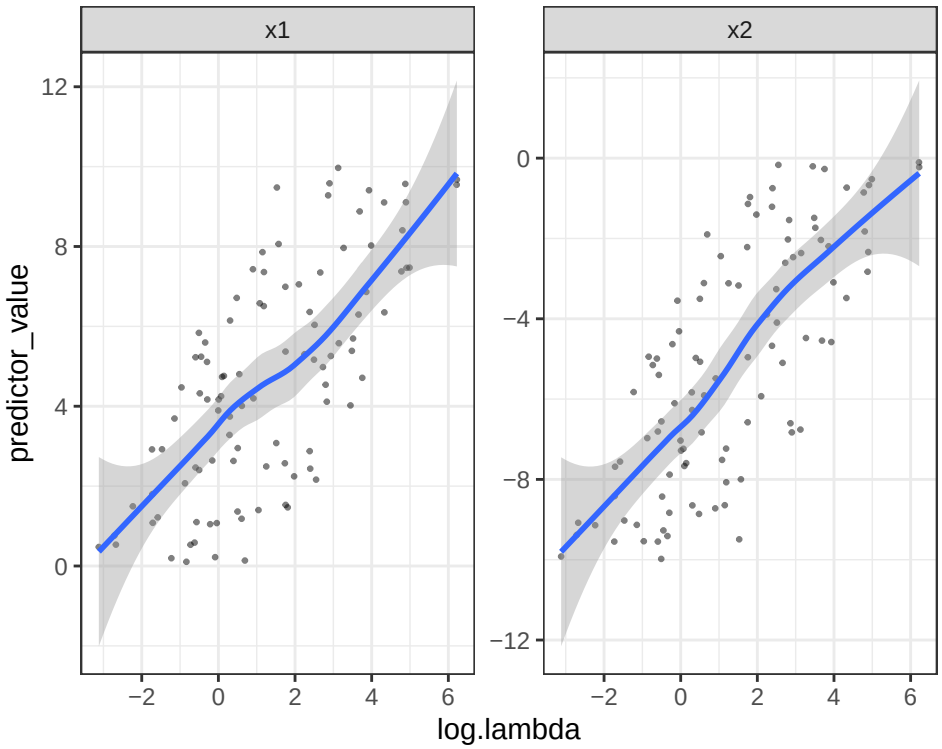
Creating fake data that has good residuals vs crabs data which has a fitted line that is not at zero. It is easy to think about why the observations may not be independent for the crabs data, if a few are observed in

clusters of units, or in certain ecologies.

```
n <- 100
x1 <- runif(n, 0, 10)
x2 <- runif(n, -10, 0)
lambda <- exp(1.5 + 0.5 * x1 + 0.5 * x2)
y <- rpois(n, lambda)
good_data <- data.frame(y, x1, x2)
log.lambda <- log(poisson_function(fn_formula = "y ~ x1 + x2", data = good_data, predict = T))
plot_data <- data.frame(log.lambda, x1, x2) |> gather(key = "predictors", value = "predictor_value", -l

ggplot(plot_data, aes(x = log.lambda, y = predictor_value)) +
  geom_point(size = 0.5, alpha = 0.5) +
  geom_smooth(method = "loess") +
  theme_bw() +
  facet_wrap(~predictors, scales = "free_y")
```

Distribution of counts follow a Poisson distribution

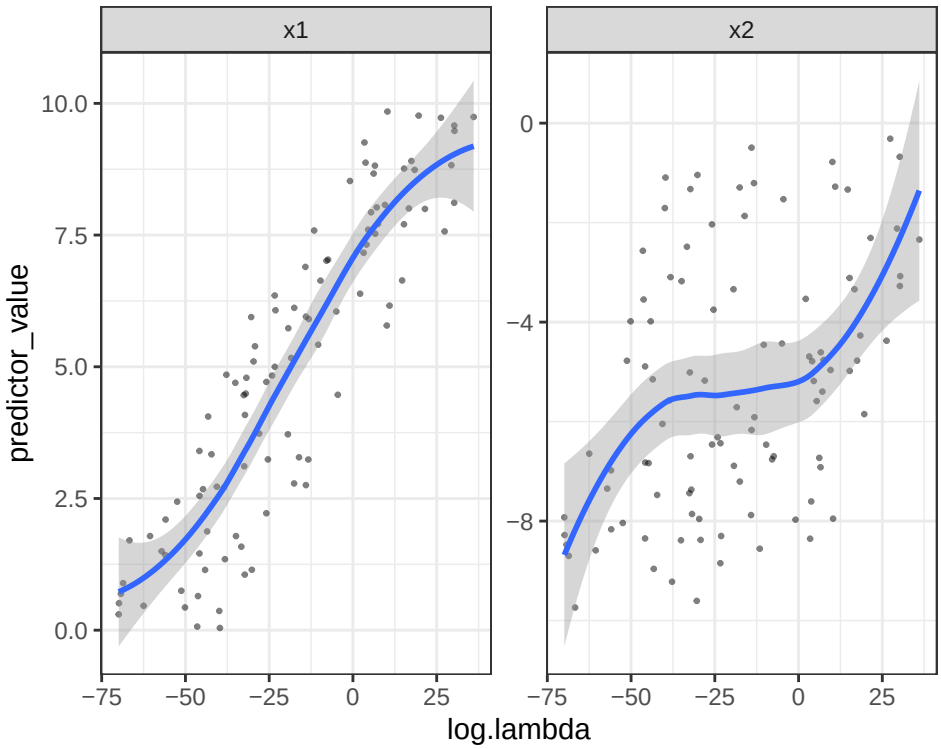
```
## `geom_smooth()` using formula = 'y ~ x'
```




```
# show non-linear data
n <- 100
x1 <- runif(n, 0, 10)
x2 <- runif(n, -10, 0)
lambda <- exp(0.5 * x1^2 + 0.5 * x1 * x2)
y <- rpois(n, lambda)
worse_data <- data.frame(y, x1, x2)
log.lambda <- log(poisson_function(fn_formula = "y ~ x1 + x2", data = worse_data, predict = T))
plot_data <- data.frame(log.lambda, x1, x2) |> gather(key = "predictors", value = "predictor_value", -l
```

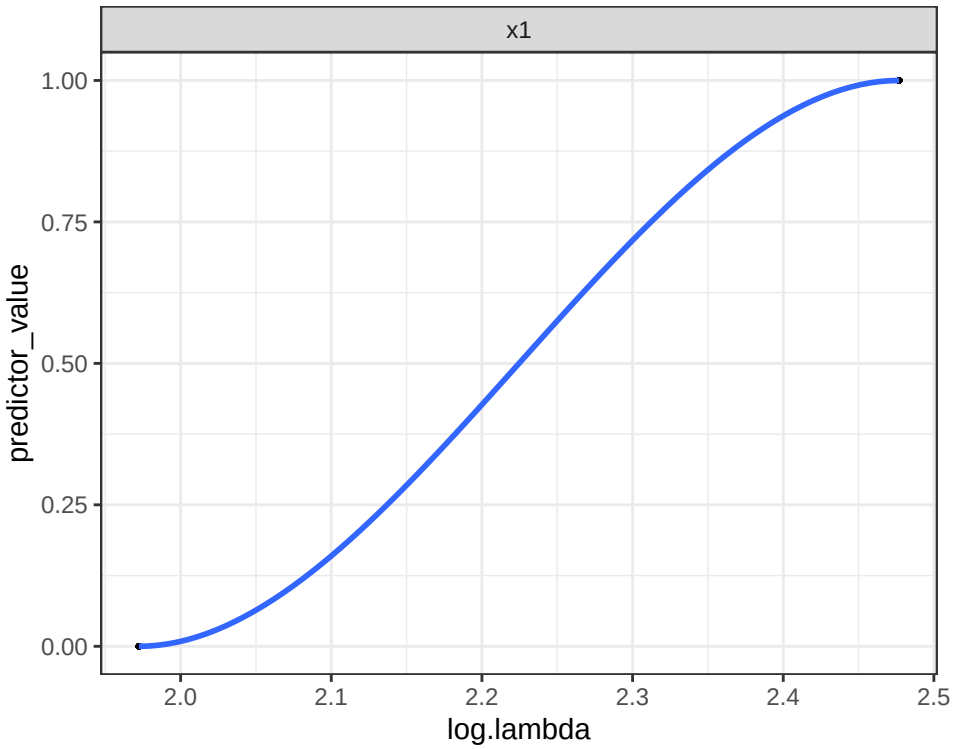
```
ggplot(plot_data, aes(x = log.lambda, y = predictor_value)) +
  geom_point(size = 0.5, alpha = 0.5) +
  geom_smooth(method = "loess") +
  theme_bw() +
  facet_wrap(~predictors, scales = "free_y")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




```
# show for binary x1
n <- 100
x1 <- sample(0:1, n, replace = T)
lambda <- exp(2 + 0.5 * x1)
y <- rpois(n, lambda)
sim_data <- data.frame(y, x1)
log.lambda <- log(poisson_function(fn_formula = "y ~ x1", data = sim_data, predict = T))
plot_data <- data.frame(log.lambda, x1) |> gather(key = "predictors", value = "predictor_value", -log.la

ggplot(plot_data, aes(x = log.lambda, y = predictor_value)) +
  geom_point(size = 0.5, alpha = 0.5) +
  geom_smooth(method = "loess") +
  theme_bw() +
  facet_wrap(~predictors, scales = "free_y")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Does the Poisson distribution predict well? We want to see a linear relationship between $log(\lambda)$ and each predictor variable. This will only be visible for continuous predictor data.

No overdispersion - the mean and variance of the distribution are equal One of the main assumptions for Poisson regression is that the mean and variance are equal. When the variance is larger than the mean, there is overdispersion. This can be formally tested using the the overdispersion parameter.

```
data(crabs, package = "glmbb")
M1 <- glm(satell ~ width, family = poisson(link = "log"), data = crabs)
M2 <- glm.nb(satell ~ width, data = crabs)
M3 <- pscl::zeroinfl(satell ~ width | width, data = crabs)
# estimate overdispersion
estimate_overdisp <- function(model_obj, data) {
  z <- resid(model_obj, type = "pearson")
  n <- nrow(data)
  k <- length(coef(model_obj))
  overdisp_ratio <- sum(z^2) / (n - k)
  p_val <- pchisq(sum(z^2), (n - k))
  return(cat("overdispersion ratio: ", overdisp_ratio, "\n", "p-value:", p_val, "\n"))
}
estimate_overdisp(M1, crabs)
```

```
## overdispersion ratio:  3.182205
##  p-value: 1
```

```
estimate_overdisp(M2, crabs)
```

```
## overdispersion ratio:  0.8464955
##  p-value: 0.07176482
```

```
estimate_overdisp(M3, crabs)
```

```
## overdispersion ratio:  1.349534
##  p-value: 0.9983357
```

The estimated overdispersion for the crabs data is 3.18, which is large, and has a p-value of 1, which indicates that the probability is essentially zero that a random variable from the distribution would be so large. When the negative binomial model is fitted, the crabs data has an estimated overdispersion of 0.85, with a smaller p-value. It is still overdispersed relative to a negative binomial distribution, but to a smaller scale than it was to a Poisson distribution.

Conclusion

Meeting the assumptions for a Poisson regression, particularly for dispersion, is quite difficult, even when simulating data. It is unclear when the mean and variance of a distribution might happen to be naturally the same.

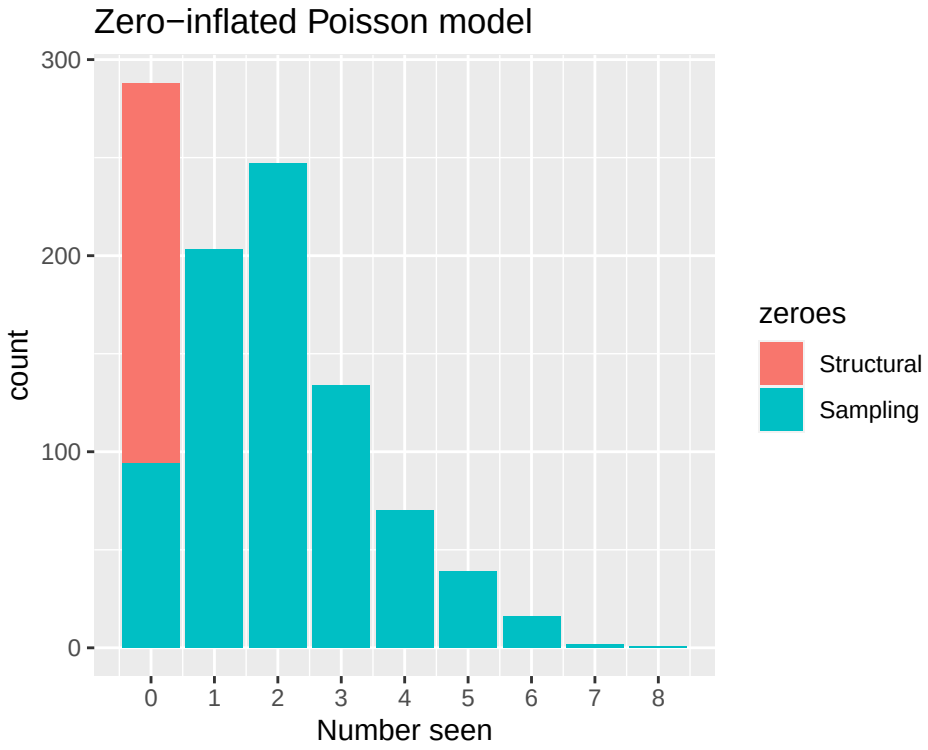
Zero-Inflated Poisson Regression

Introduction

Poisson regression is used for count and rate data, but if may not be the best model when there are excess zeroes in that data, which is when we may use Zero-inflated poisson regression instead. ZIP models have one parameter representing the probability of a structured zero, and another representing the Poisson mean. The ZIP distribution has the parameters π and λ, denoted by $ZIP(\pi, \lambda)$, with this probability mass function.

$$P(X = k) = \begin{cases} \pi + (1 - \pi)(exp(-\lambda)) & \text{if } k = 0 \\ (1 - \pi)exp(-\lambda)\frac{\lambda^k}{k!} & \text{if } k = 1, 2, 3, ... \end{cases} \tag{1}$$

The figure below shows a zero-inflated Poisson model, where the zeros are either sampling zeros or structural zeros.



Uses

An example of when ZIP-distributed count happens is when ecologists counting plants or animals get a zero when the species is absent at many sites, but get a Poisson distributed count when they are present. Another example is for estimating the the dental health of individuals, by counting how many dental cavities there are. Most people have 0 dental cavities as children, so this is a good use case for ZIP.

Assumptions

- It follows the same assumptions for the Poisson regression for the counts generated by the Poisson process, and assumptions that apply for the logistic model that models the probability of being a zero.

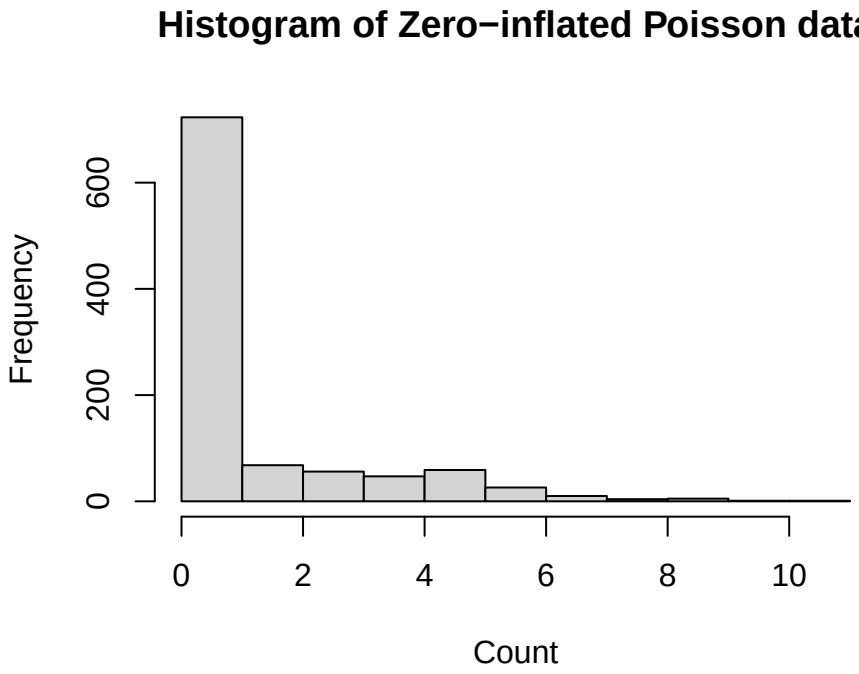
Our Zero-inflated Poisson Regression Implementation

```
zippoisson_function <- function(fn_formula, data) {
  number_omitted <- nrow(data) - nrow(na.omit(data))
  data <- na.omit(data)

  vars <- all.vars(as.formula(fn_formula))
  y_name <- vars[1]
  covL <- data[, vars[2]]
  covp <- data[, vars[3]]
  n <- nrow(data)
  Y <- matrix(data[, y_name], nrow = n, ncol = 1)

  optim_zip <- function(beta) {
    lambda <- exp(beta[1] + beta[2] * covL)
    p <- plogis(beta[3] + beta[4] * covp)
    lik <- p * (Y == 0) + (1 - p) * dpois(Y, lambda)
    return(-sum(log(lik)))
  }
  result <- optim(par = rep(0, 4), fn = optim_zip, hessian = T)
  OI <- solve(result$hessian)
  se <- sqrt(diag(OI))
  z_value <- result$par / se
  p_value <- 2 * pnorm(-1 * abs(z_value))

  coef <- rbind(result$par, se, z_value, p_value)

  colnames(coef) <- c("(Intercept)", vars[2], "(Intercept)", vars[3])
  rownames(coef) <- c("Estimate", "Std. Error", "z value", "p value")
  return(t(coef))
}
```

Comparing performance

```
n <- 1000
covL <- seq(0, 1, length.out = n)
covp <- seq(0, 1, length.out = n)
trueMeans <- exp(1.5 - 0.5 * covL)
probability <- plogis(-0.5 + 2.5 * covp)
U <- runif(n, 0, 1)
y <- rpois(n, trueMeans)
y[U < probability] <- 0
zip_data <- data.frame(y, covL, covp)
```

```r
hist(zip_data$y, main = "Histogram of Zero-inflated Poisson data", xlab = "Count")
```




```r
# comparing performance of our implementation with zeroinfl
zippoisson_function(fn_formula = "y ~ covL | covp", data = zip_data)
```

```
##               Estimate Std. Error   z value       p value
## (Intercept)  1.4782770  0.0493991 29.925183 9.256989e-197
## covL        -0.6141920  0.1264138 -4.858583  1.182287e-06
## (Intercept) -0.5216485  0.1380291 -3.779265  1.572918e-04
## covp         2.7681417  0.2793013  9.910952  3.730730e-23
```

```r
summary(pscl::zeroinfl(y ~ covL | covp))$coef
```

```
## $count
##               Estimate Std. Error   z value      Pr(>|z|)
## (Intercept)  1.4784156 0.04939646 29.929585 8.113008e-197
## covL        -0.6143214 0.12641067 -4.859728  1.175472e-06
##
## $zero
##               Estimate Std. Error   z value     Pr(>|z|)
## (Intercept) -0.5220753  0.1380288 -3.782364 1.553460e-04
## covp         2.7683227  0.2792894  9.912023 3.690974e-23
```

Checking Assumptions

Suitability for ZIP or Poisson Using the crabs data again, we will check the suitability for ZIP or Poisson with a likelihood ratio test.

```r
zip_model <- pscl::zeroinfl(satell ~ width | width, data = crabs)
pois_model <- glm(satell ~ width, family = poisson, data = crabs)
lmtest::lrtest(zip_model, pois_model)
```

```
## Warning in modelUpdate(objects[[i - 1]], objects[[i]]): original model was of
## class "zeroinfl", updated model is of class "glm"
```

```
## Likelihood ratio test
##
## Model 1: satell ~ width | width
## Model 2: satell ~ width
##   #Df  LogLik Df  Chisq Pr(>Chisq)
## 1   4 -364.82
## 2   2 -461.59 -2 193.53  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

If we set the significance level at 0.05, the likelihood ratio test shows that we should reject the null hypothesis, so the ZIP model offers an improvement in fit over the Poisson model, since the ZIP model has a log likelihood closer to zero.

```
pois_model <- glm(y ~ covL + covp, data = zip_data)
zip_model <- pscl::zeroinfl(y ~ covL | covp, data = zip_data)
lmtest::lrtest(zip_model, pois_model)
```

```
## Warning in modelUpdate(objects[[i - 1]], objects[[i]]): original model was of
## class "zeroinfl", updated model is of class "glm"
```

```
## Likelihood ratio test
##
## Model 1: y ~ covL | covp
## Model 2: y ~ covL + covp
##   #Df  LogLik Df  Chisq Pr(>Chisq)
## 1   4 -1161.3
## 2   3 -2045.5 -1 1768.4  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This likelihood ratio test of our own constructed dataset following a ZIP distribution also has a p-value of smaller than 0.05, and that the ZIP model has a log likelihood closer to zero, indicating better suitability for a ZIP model.

Conclusion

ZIP models provide a flexible framework by combining a Poisson distribution for positive counts with a logistic regression component to model the excess zeros. It is most appropriate to plot the data and see whether a dataset containing a count response variable is suited for Poisson or ZIP, and then conducting a likelihood ratio test to compare them.

A Note

In determining how appropriate a particular regression model is for a problem, we think it might be valuable to move beyond a 'assumption #1 met' or 'assumption #1 not met' assessment. Instead, it might be more helpful for researchers to explore their data, and test assumptions on their own to determine 'what level of assumption-violation' might they be willing to accept given they are experts in their fields and have domain-specific contextual knowledge. This is not to say that assumptions can be violated without consequences.

Also, it might be difficult to determine the thresholds for having actually met or not met an assumption. Thus, a function which simply outputs a 'yes' or 'no' to whether assumptions for a particular regression implementation have been met did not seem appropriate. We therefore, test assumptions by intentionally breaking them and noting the extent of bias-ness that they result.