DIY Regression

Jasmine Siswandjo and Saumya Seth

2023-12-17

```
set.seed(123)

library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(dplyr)
library(MASS)
```

```
##
## Attaching package: 'MASS'
##
## The following object is masked from 'package:dplyr':
##
##     select
```

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

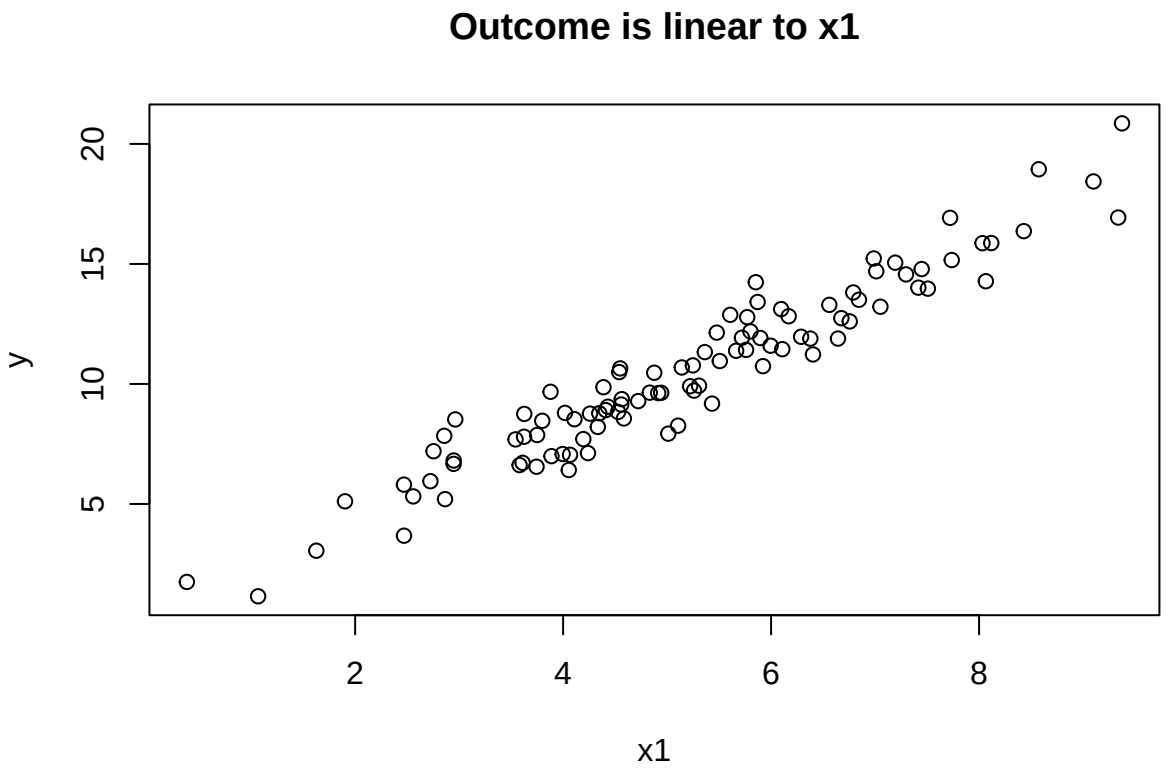
Our Linear Regression implementation: (Note that we use bootstrapping to estimate standard errors)

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

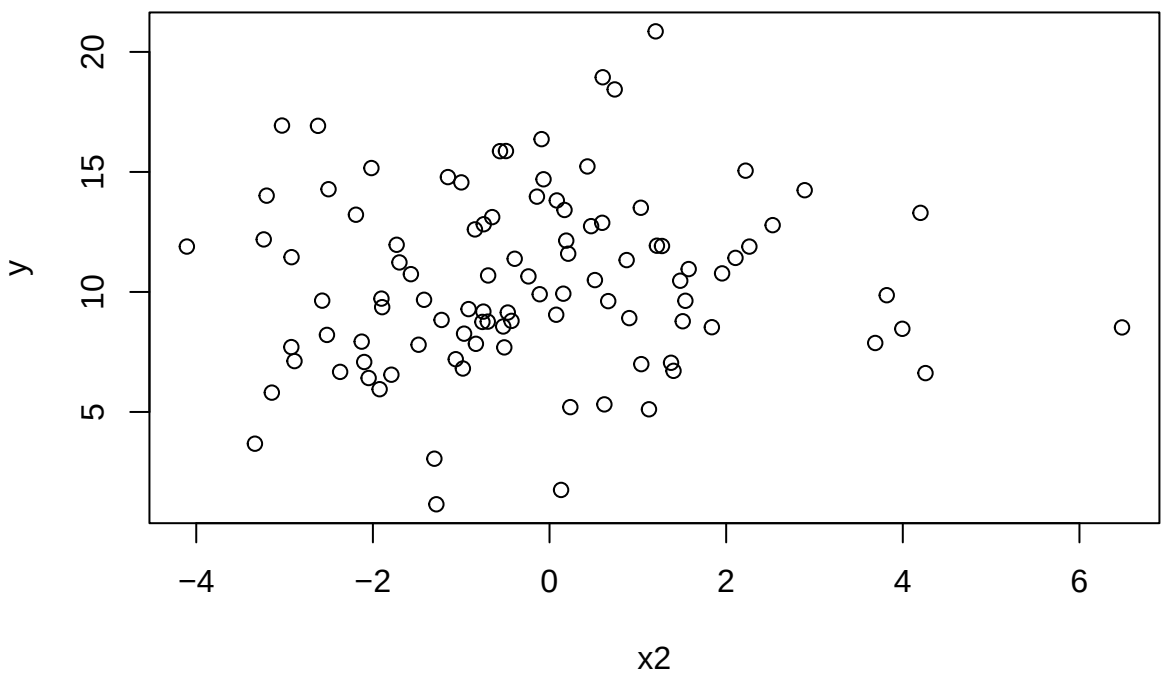
```
  p <- 2 * pt(t, df, lower = F)
  # calculating the residuals
  residual <- sqrt(mean((Y - X %*% beta)^2))
  # defining the row names of the output data frame
  rownames <- c()
  for (i in 1:((ncol(X)) - 1)) {
    rownames[i] <- i
  }
  # returning a data frame akin to the lm output
  return(
    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))
    )
  )
}
```

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

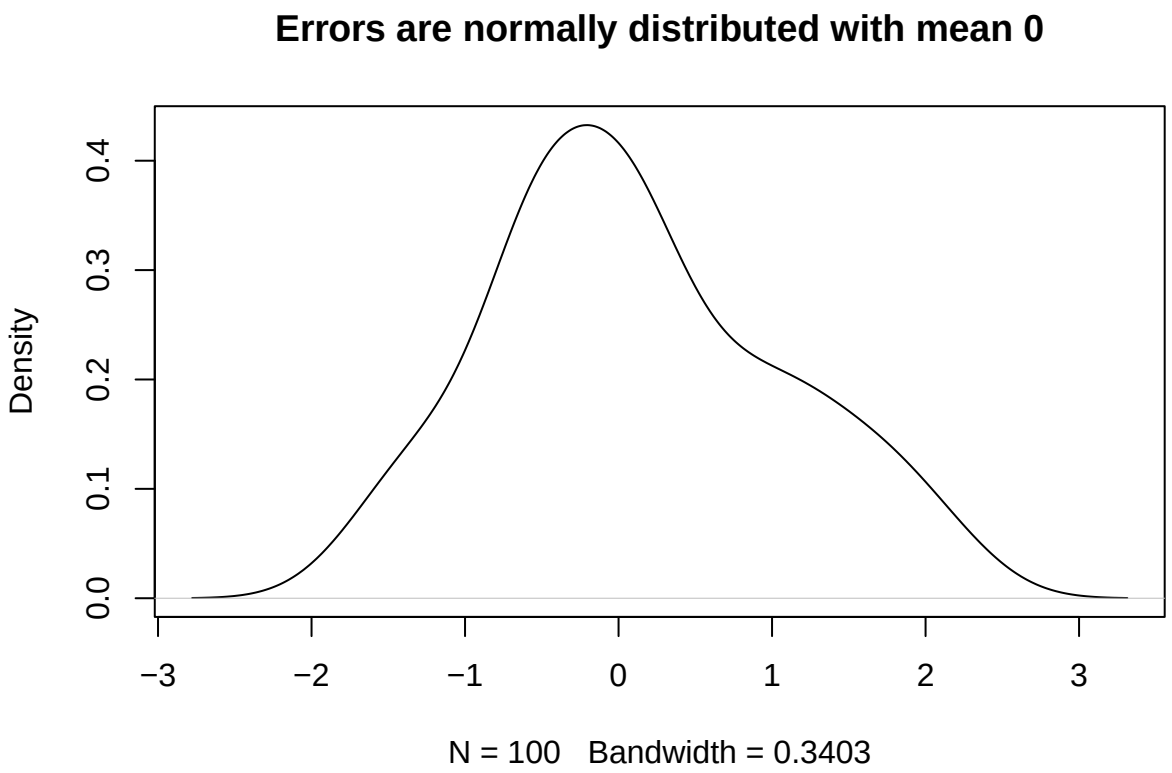
```
test_linear_regression_data <-
  data.frame(
    x1 = rnorm(100, mean = 5, sd = 2),
    x2 = rnorm(100, mean = 0, sd = 2)
  )
error <- rnorm(100, mean = 0, sd = 1) # errors are homoscedastic
test_linear_regression_data$y <-
  2 * test_linear_regression_data$x1 +
  0.2 * test_linear_regression_data$x2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

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



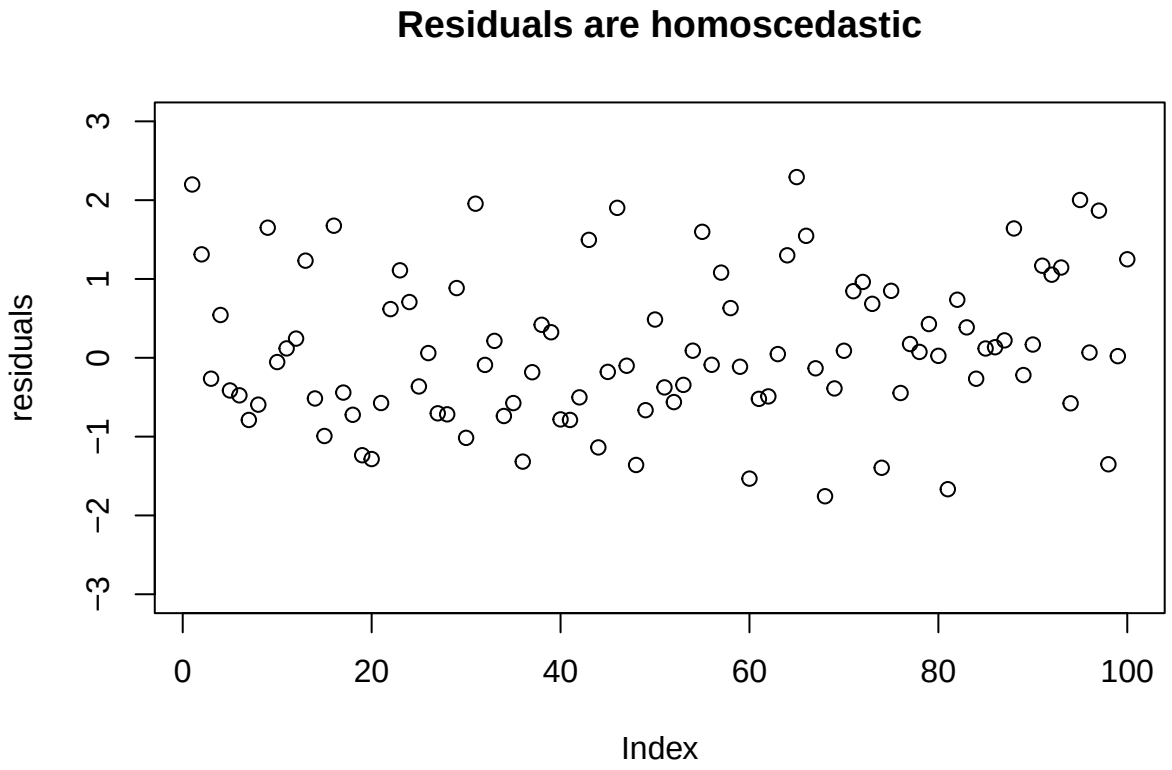

```
plot(test_linear_regression_data$x2, test_linear_regression_data$y,
  xlab = "x2", ylab = "y",
  main = "Outcome is linear to x2 (it is not apparent in this plot but our data structure captures this
)
```



```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)
)
```



Applying the function we created on this data set.

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

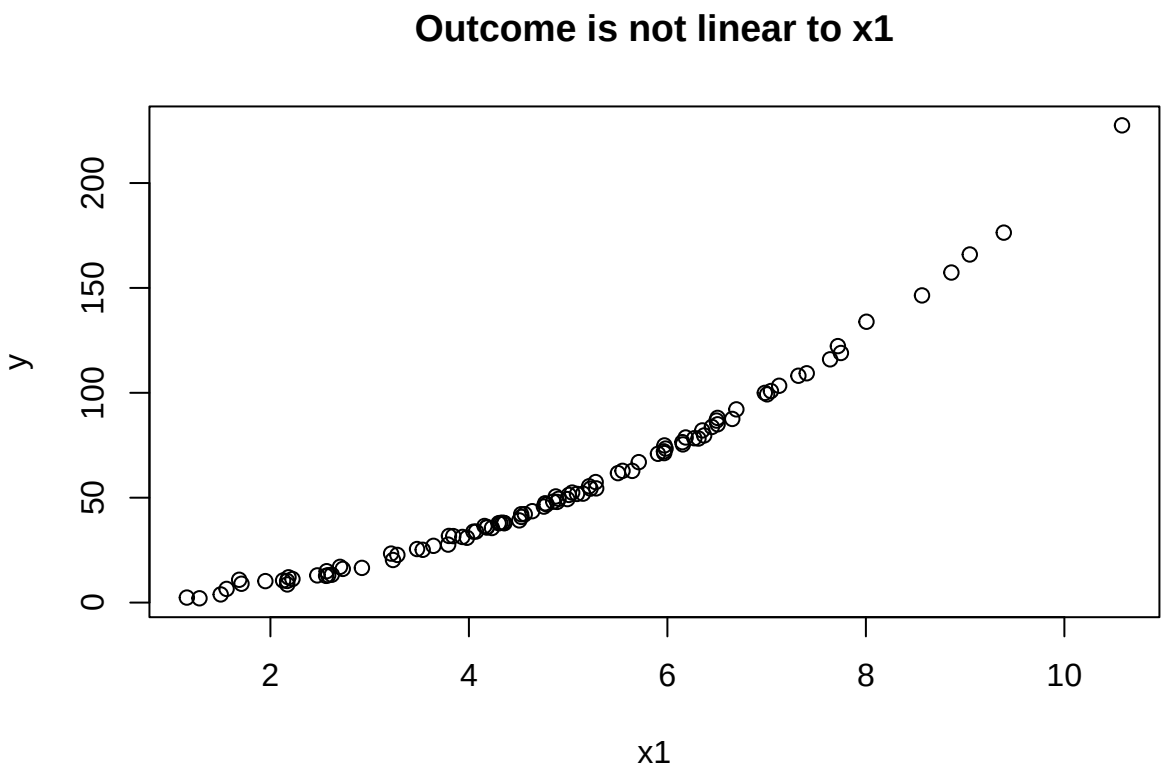
```
## [1] 0.9369198
```

Breaking Assumptions

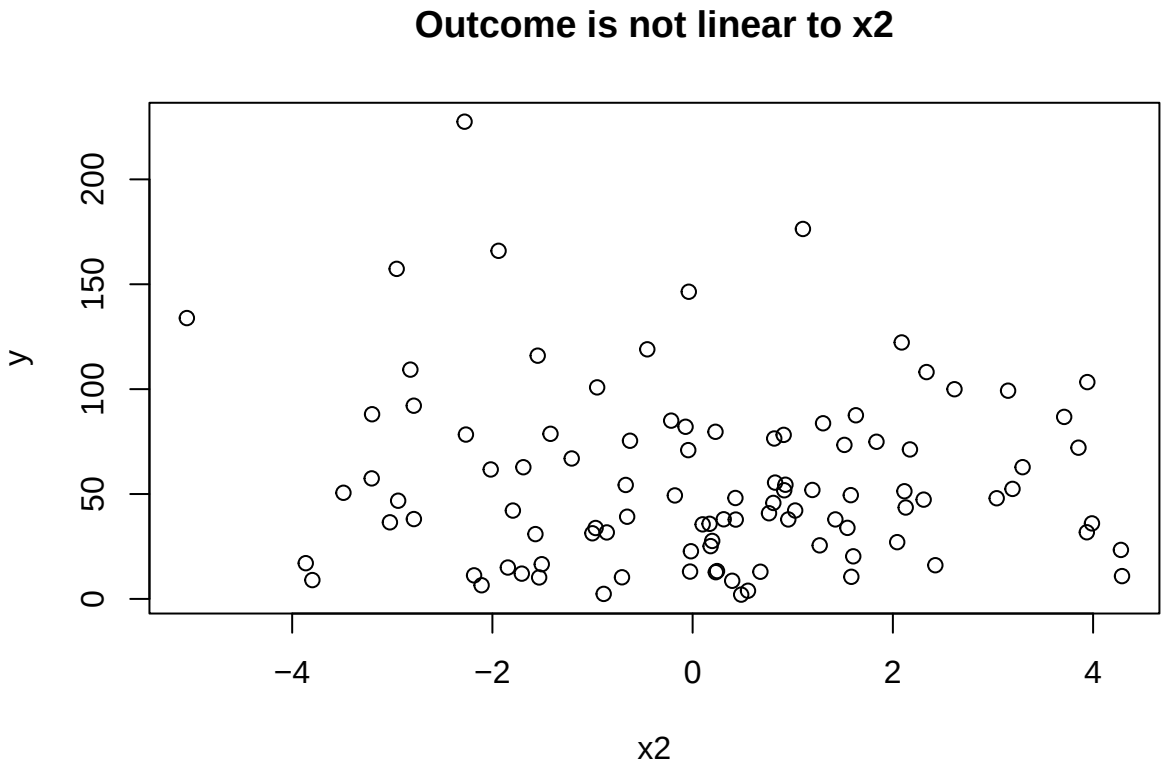
Breaking the assumption of the predictors and outcome following a linear relationship

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
)
our_implementation_not_linear
```

```
##              Estimate Std.Error   t.value      p.value Residual DegOfFreedom
## (Intercept) -44.28663 4.7210670 -9.380640 2.000000e+00 10.22817           97
## x1           20.61334 1.0238831 20.132512 2.060350e-36       NA           NA
## x2           -1.10814 0.5220249 -2.122773 1.963681e+00       NA           NA
```

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

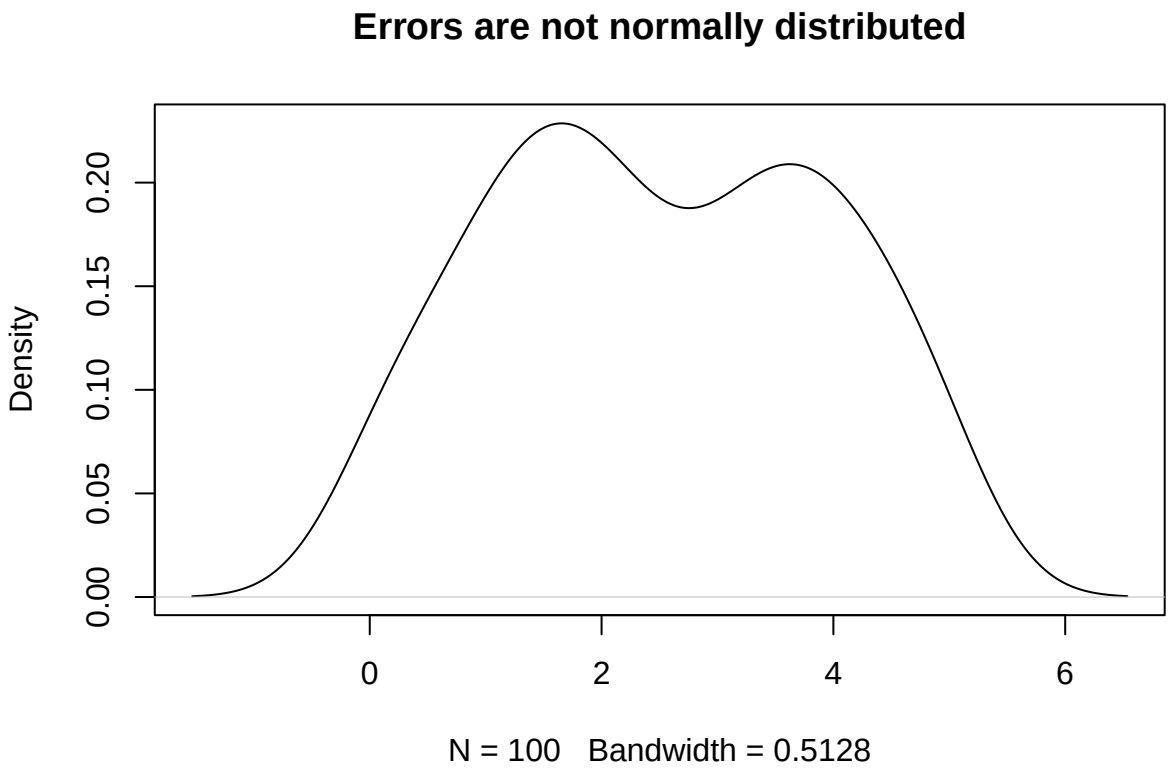
```
## [1] 10.22817
```

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
)
our_implementation_not_normally_dist
```

```
##               Estimate  Std.Error   t.value      p.value Residual DegOfFreedom
## (Intercept) 2.3127526 0.40399235  5.724743 1.156540e-07 1.414274           97
## x1          2.0345203 0.07612338 26.726615 1.575099e-46       NA           NA
## x2          0.2800552 0.07939081  3.527552 6.426545e-04       NA           NA
```

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

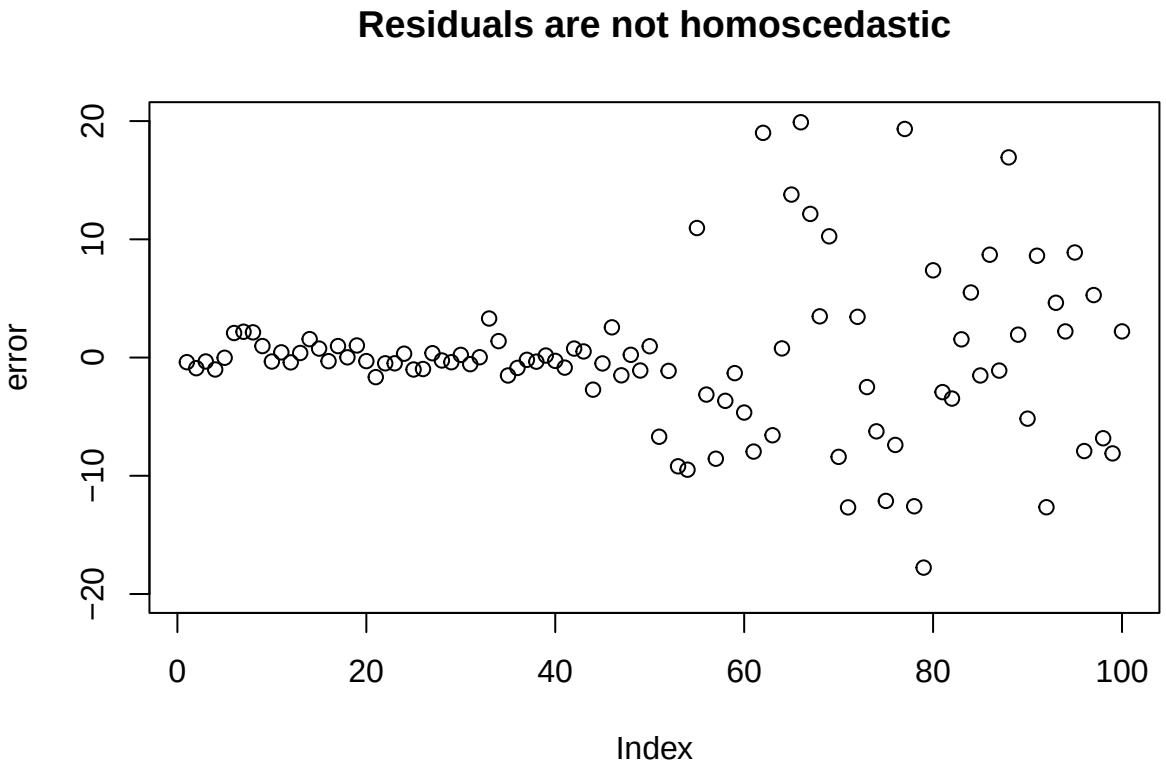
```
## [1] 1.414274
```

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
)
our_implementation_not_homoscedastic
```

```
##               Estimate Std.Error    t.value      p.value Residual DegOfFreedom
## (Intercept) -1.2104181 1.4231532 -0.8505184 1.602868e+00 6.419781           97
```

```
## x1           2.2465795 0.2597406  8.6493199 1.098883e-13       NA           NA
## x2           0.6025916 0.2761419  2.1821807 3.151087e-02       NA           NA
```

```
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)
  }
  # 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
```

```r
# bootstrapping to estimate the standard errors
num_bootstraps <- 10000
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
}
# returning a data frame akin to the glm probit output
return(
  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))
```

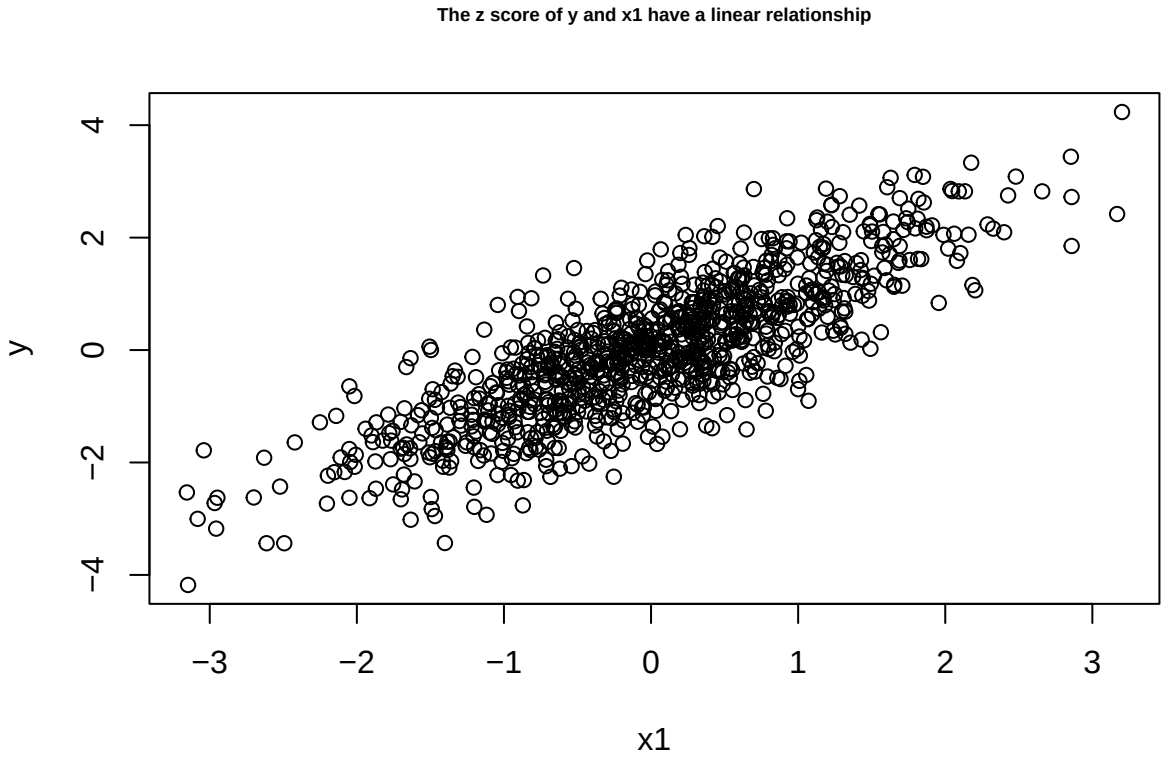
```
    )
  )
}
```

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

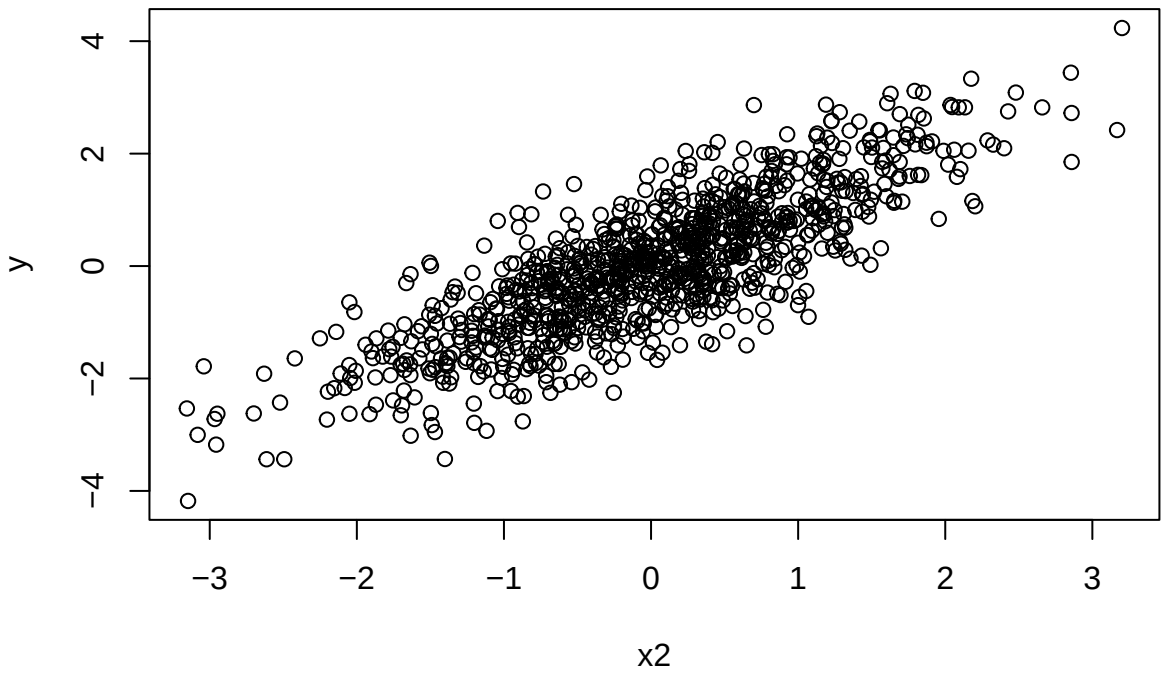
```
predict_probit <-
  function(data, ..., y, implementation_probit) {
    n <-
      implementation_probit$DegOfFreedom[1] + nrow(implementation_probit)
    input_covariate_values <- c(...)
    X <-
      matrix(
        c(rep(1, n), input_covariate_values),
        nrow = n,
        ncol = nrow(implementation_probit)
      )
    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.

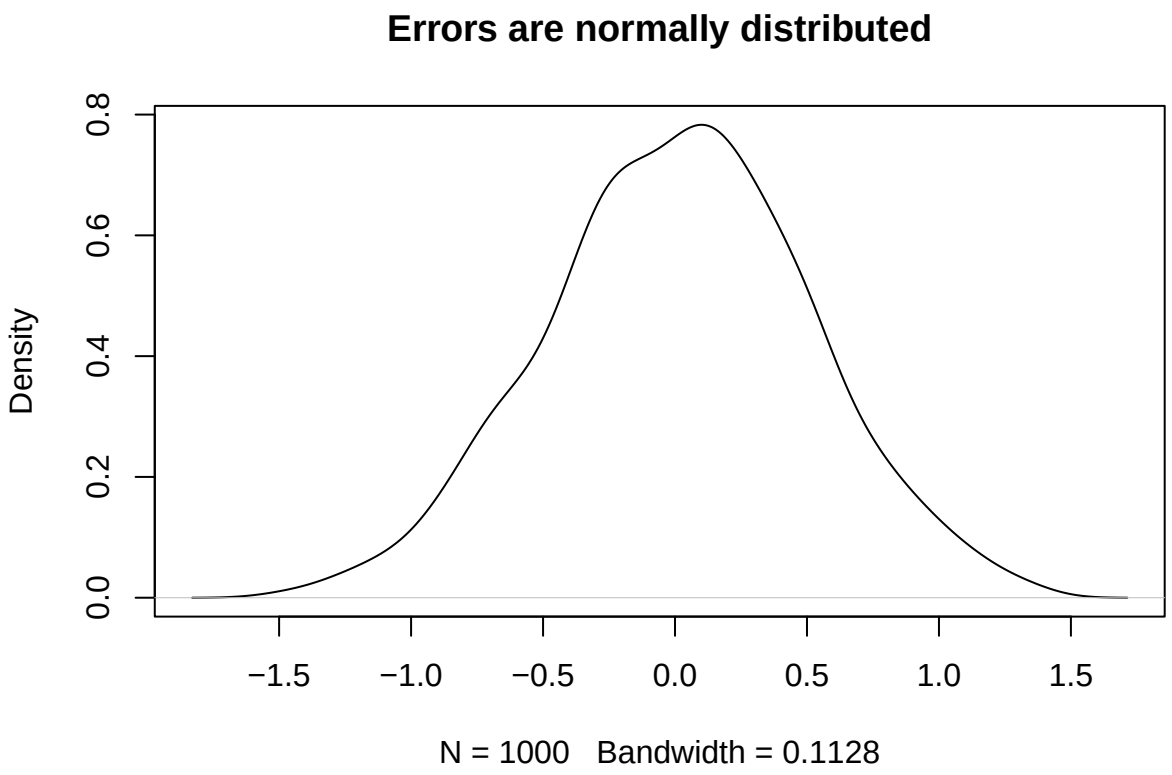
```
test_probit_regression_data <- data.frame(
  x1 = rnorm(1000, 0, 1),
  x2 = rnorm(1000, 0, 1)
)
error <- rnorm(1000, mean = 0, sd = 0.5)
test_probit_regression_data$y <- test_probit_regression_data$x1 +
  0.5 * test_probit_regression_data$x2 +
  error
test_probit_regression_data$y <-
  qnorm(pnorm(test_probit_regression_data$y))

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"
)
```



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

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



Applying the function we created on this data set.

```r
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
  )
our_implementation_probit
```

```
##                Estimate  Std.Error    z.value       p.value DegOfFreedom
## (Intercept) 0.008642326 0.03997910  0.2161711  8.288544e-01          997
## x1          1.861391305 0.03989830 46.6533950  0.000000e+00           NA
## x2          0.944419878 0.03989357 23.6734877 6.764030e-124           NA
```

Comparing our output to R's output.

```r
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
```

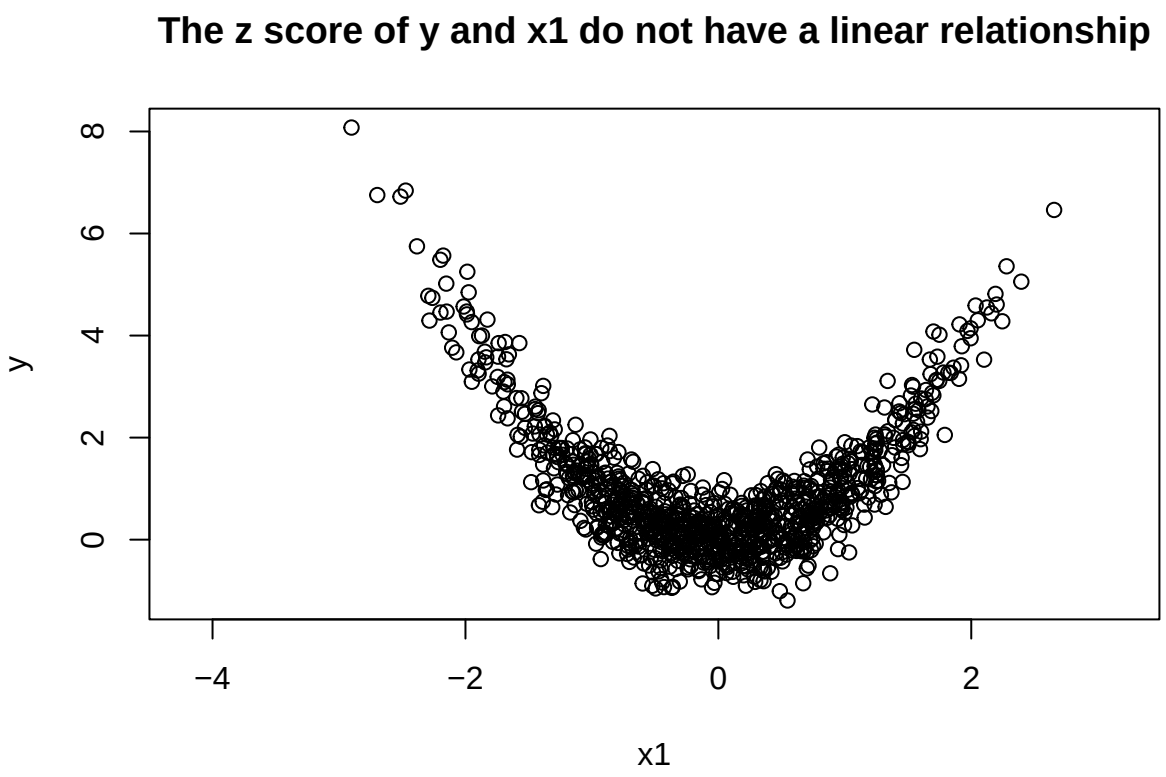
```
## [1] 0.852
```

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.

```r
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))

plot(test_probit_regression_data_not_linear$x1, test_probit_regression_data_not_linear$y,
  main = "The z score of y and x1 do 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
  )
our_implementation_probit_not_linear
```

```
##                Estimate  Std.Error    z.value      p.value DegOfFreedom
## (Intercept)  0.79830122 0.04545988 17.5605641 4.938201e-69          997
## x1          -0.04015593 0.04710952 -0.8523953 1.606005e+00           NA
## x2          -0.02160582 0.04351634 -0.4964991 1.380458e+00           NA
```

```
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
```

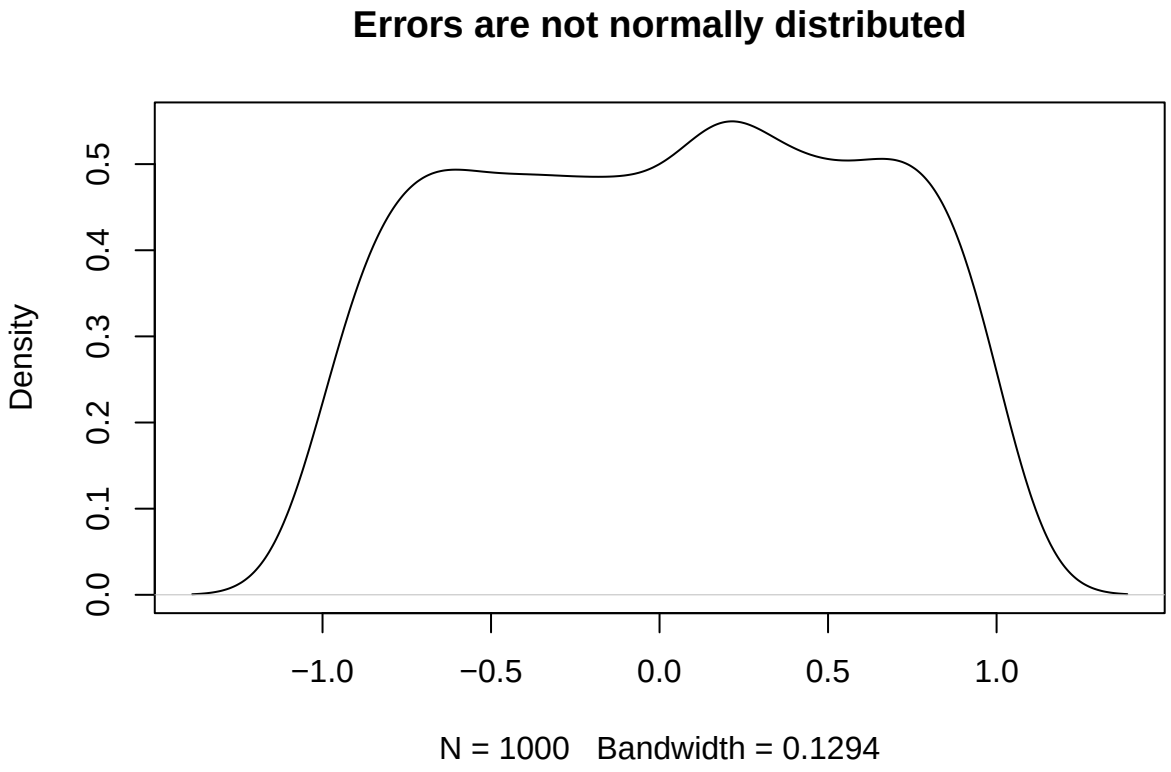
```
## [1] 0.788
```

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))

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




```
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.

```
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
  )
our_implementation_probit_not_normally_dist
```

```
##               Estimate  Std.Error   z.value      p.value DegOfFreedom
## (Intercept) 0.13288527 0.03979900  3.338910 0.0008410779          997
## x1          1.75679367 0.04061130 43.258738 0.0000000000           NA
## x2          0.07675678 0.04131399  1.857888 0.0631848576           NA
```

```
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] 0.824
```

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    0.788
## Normality assumption violated    0.824
```

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
  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 <- 10000
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
}
# returning a data frame akin to the glm probit output
return(
  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))
    )
  )
}
```

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

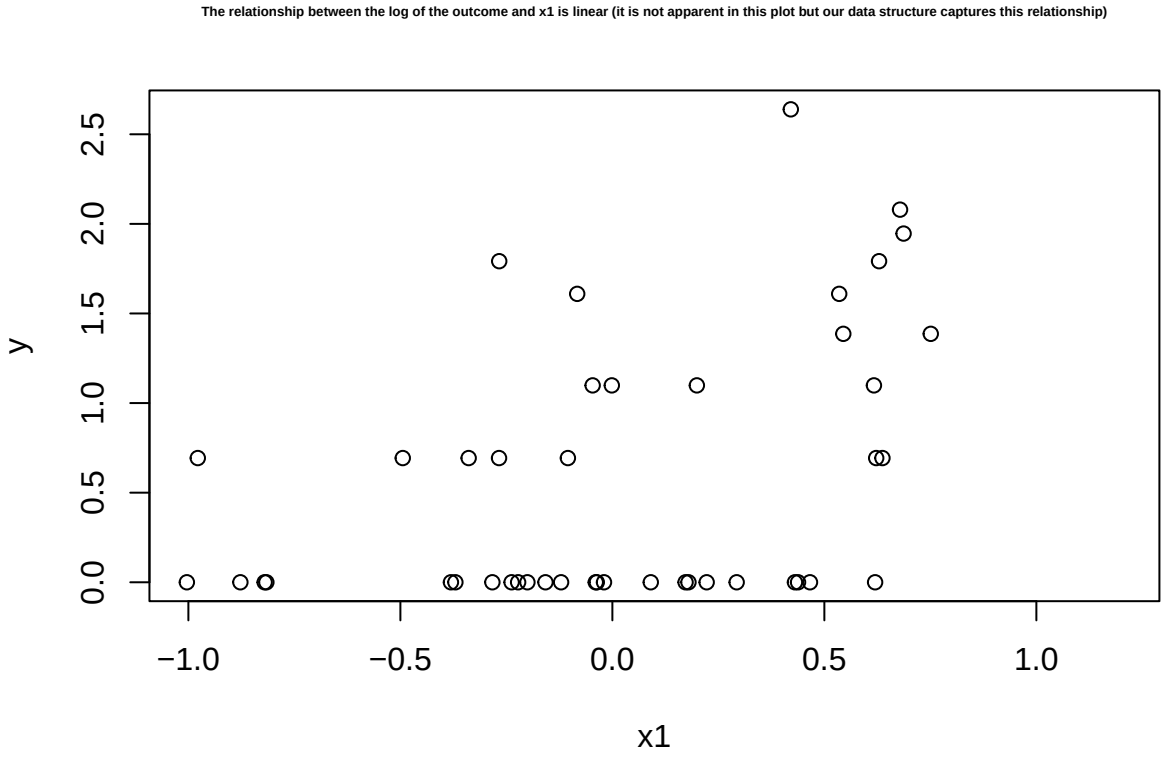
```
predict_neg_binom <-
  function(data, ..., y, implementation_neg_binom) {
    n <-
      implementation_neg_binom$DegOfFreedom[1] + nrow(implementation_neg_binom)
    input_covariate_values <- c(...)
    X <-
      matrix(
        c(rep(1, n), input_covariate_values),
        nrow = n,
        ncol = 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.

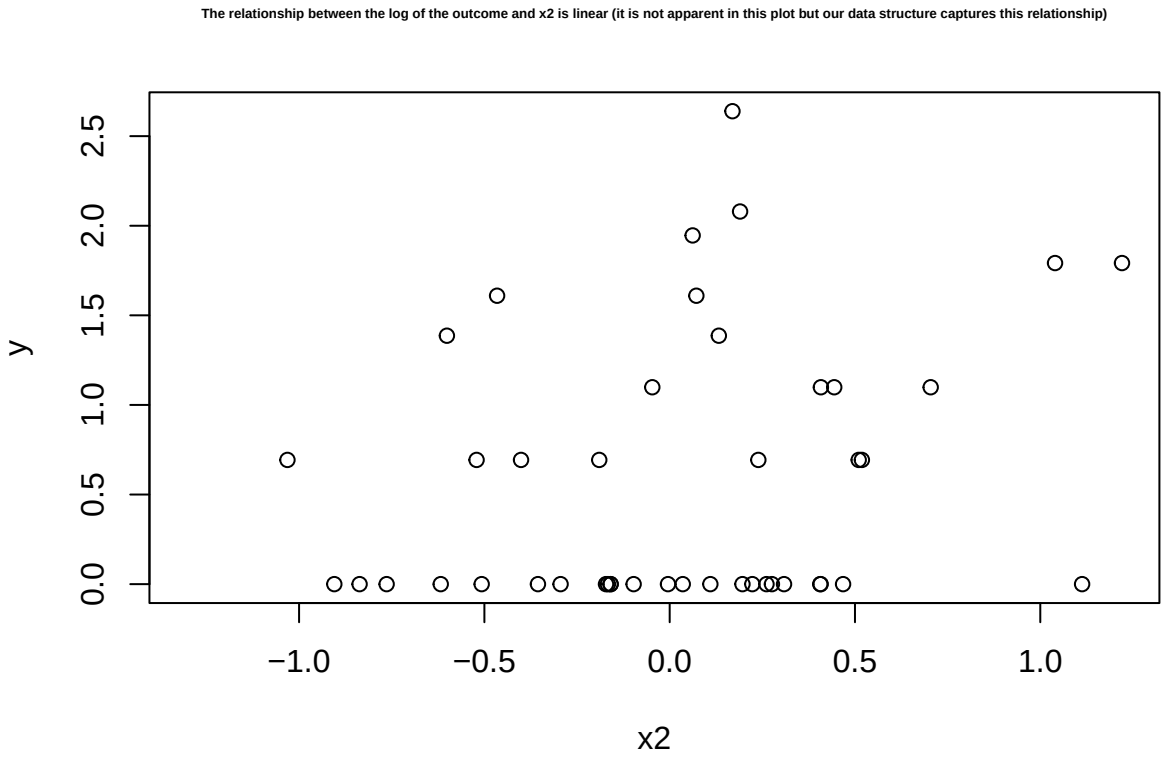
```
x1 <- rnorm(100, mean = 0, sd = 0.5)
x2 <- rnorm(100, mean = 0, sd = 0.5)
y <- rnbinom(100, mu = exp(x1 + x2), size = 0.5)
test_neg_binom_regression_data <- data.frame(x1, x2, y)
# to ensure that the variance of the outcome variable is greater
# than its mean
var(y) > mean(y)
```

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

```
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"
)
```




```
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
)
```



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
  )
our_implementation_neg_binom
```

```
##                Estimate Std.Error    z.value    p.value DegOfFreedom
## (Intercept) -0.05793752 0.1969158 -0.2942249 1.23141394           97
## x1           1.01524367 0.3960700  2.5632934 0.01036844           NA
## x2           0.77721158 0.3642657  2.1336392 0.03287233           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 = 0.6415721834, link = log)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6298  -1.0581  -0.8142   0.2682   2.3723
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.05538    0.16692  -0.332  0.74007
## x1           0.99304    0.34739   2.859  0.00426 **
## x2           0.76070    0.32490   2.341  0.01921 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(0.6416) family taken to be 1)
##
##     Null deviance: 103.379  on 99  degrees of freedom
## Residual deviance:  88.624  on 97  degrees of freedom
## AIC: 276.05
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  0.642
##           Std. Err.:  0.191
##
##  2 x log-likelihood:  -268.046
```

We note that the results are similar.

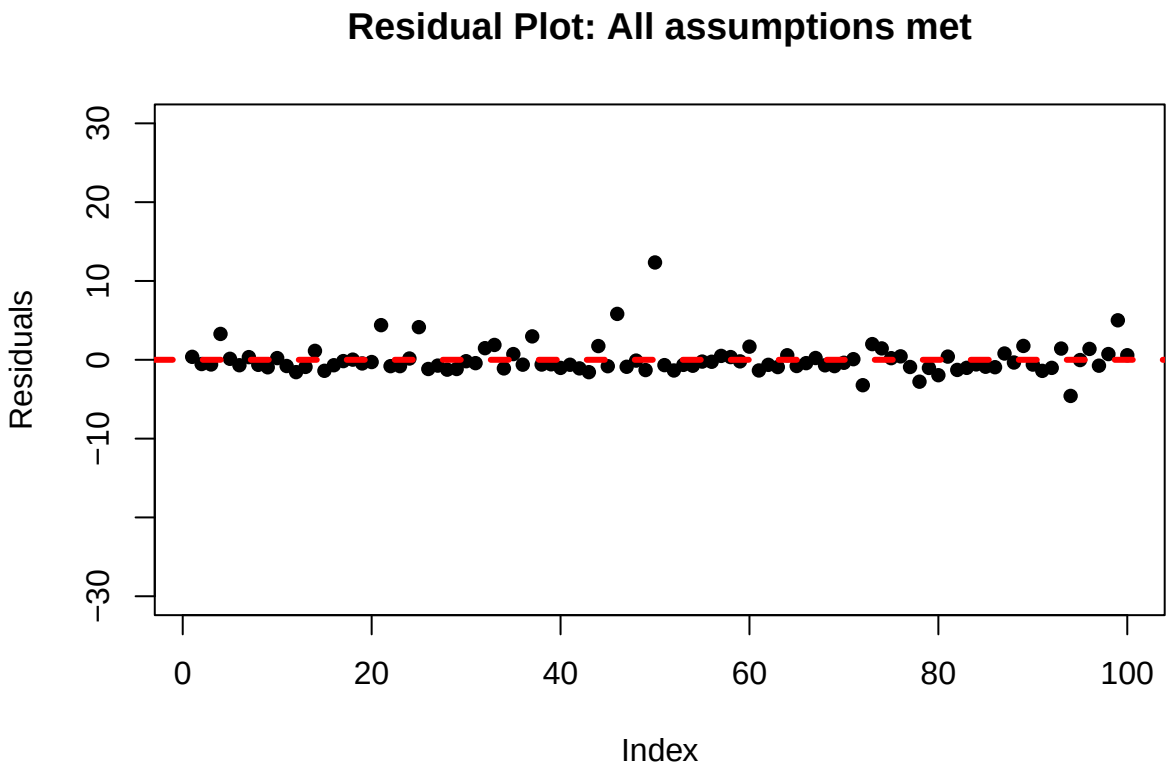
We followed all assumptions of Negative Binomial Regression in regressing y on x1 and x2 using the test_neg_binom_regression_data data set. We will compare the residual of this regression to that of all the others where assumptions will be broken.

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] 1.962647
```

```
# 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

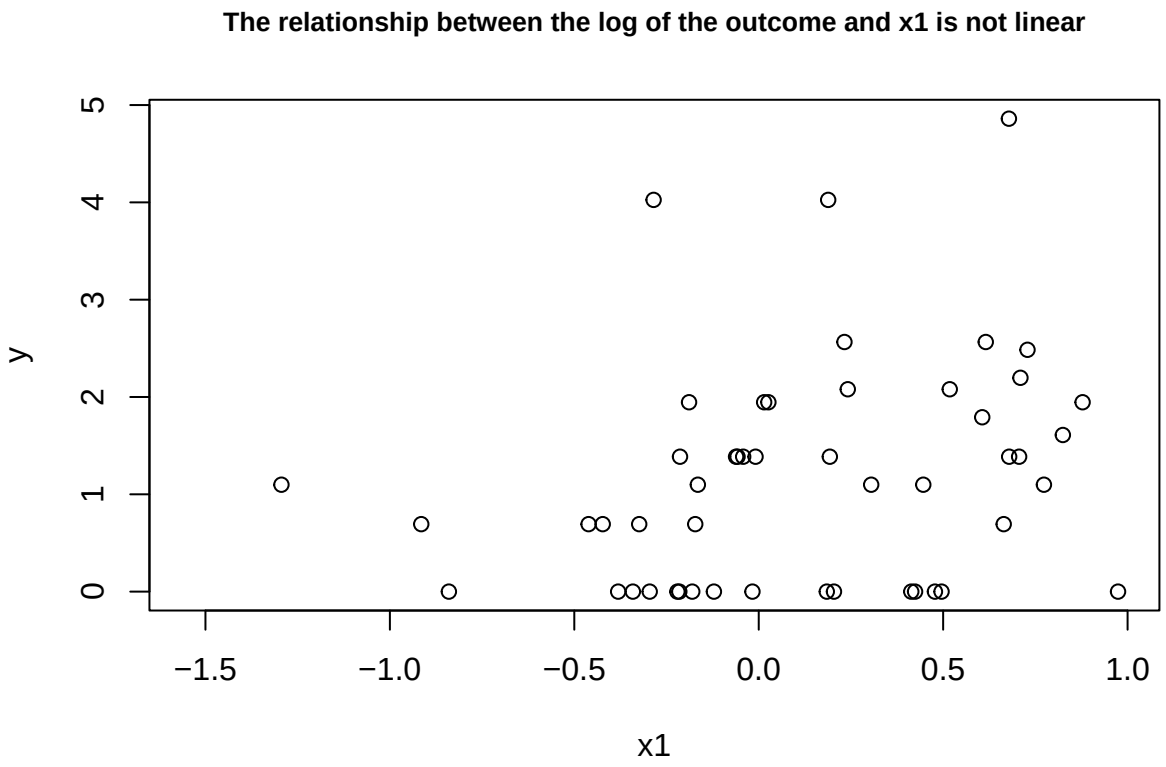
Breaking the assumption that the relationship between the predictors and the log of the outcome's mean is linear

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

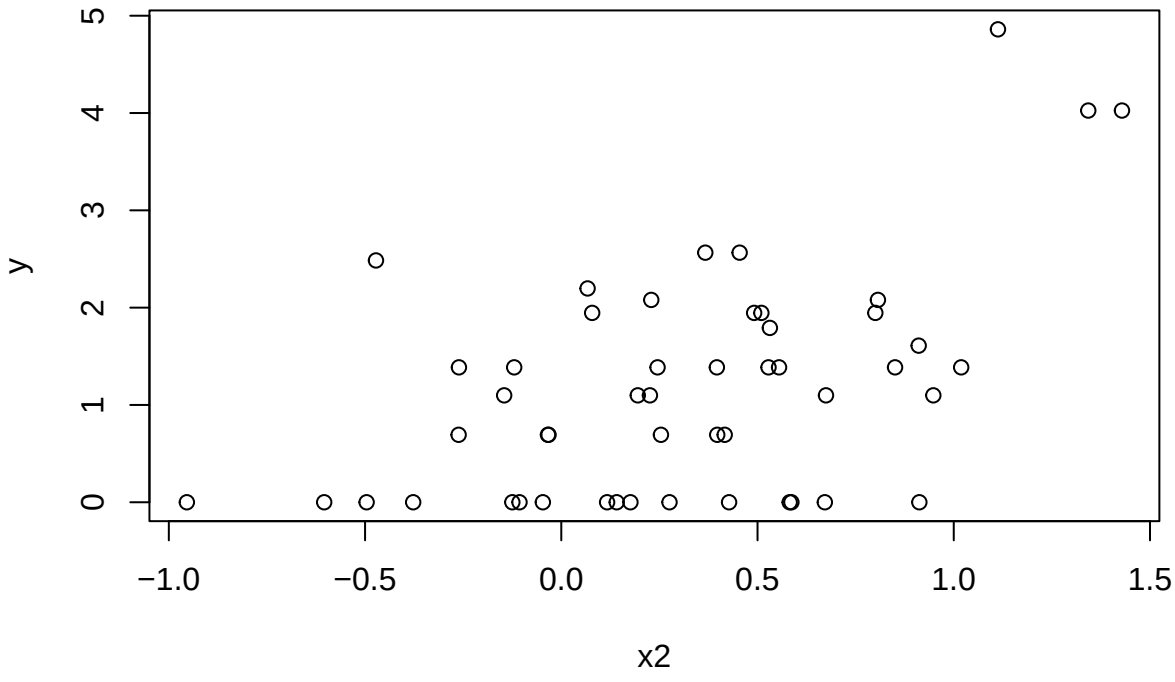
```
x1 <- rnorm(100, mean = 0, sd = 0.5)
x2 <- rnorm(100, mean = 0, sd = 0.5)
y <- rnbinom(100, mu = exp(x1 + x2)^2, size = 0.5)
test_neg_binom_regression_data_not_linear <- data.frame(x1, x2, y)
# to ensure that the variance of the outcome variable is greater
# than its mean
var(y) > mean(y)
```

```
## [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"
)
```




```
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"
)
```



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

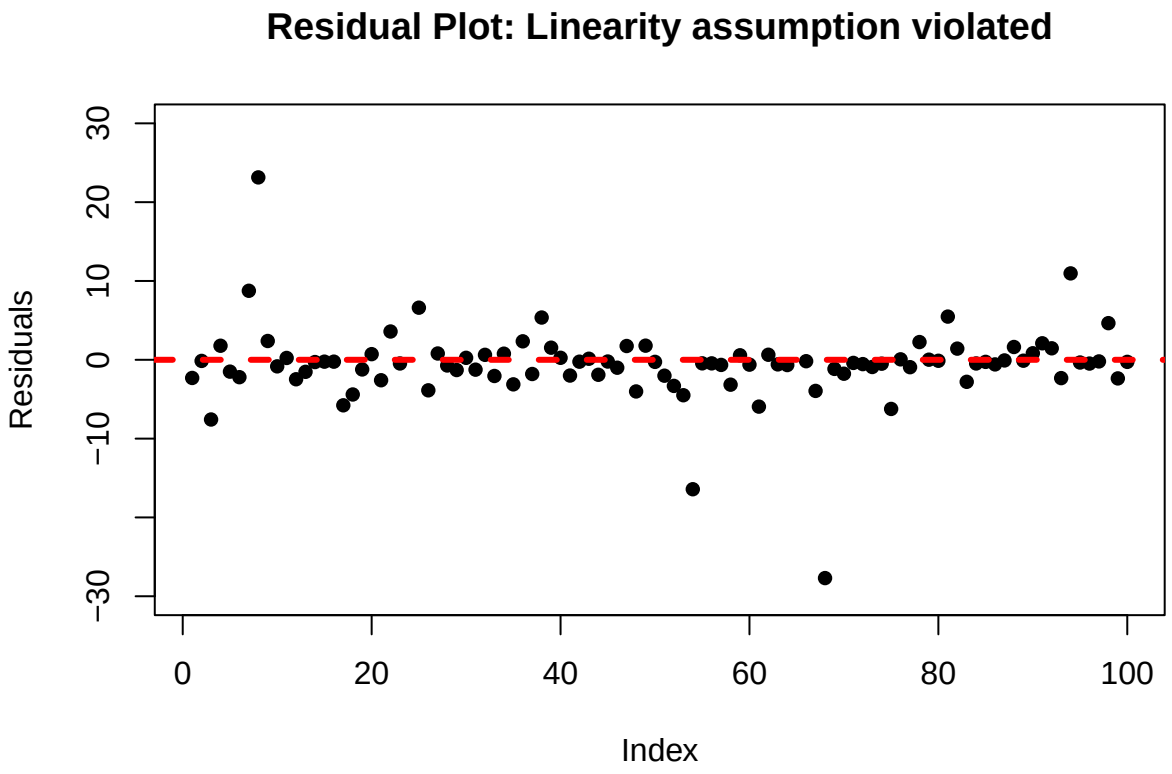
```
our_implementation_neg_binom_not_linear <-
  negative_binomial_regression(
    test_neg_binom_regression_data_not_linear,
    test_neg_binom_regression_data_not_linear$x1,
    test_neg_binom_regression_data_not_linear$x2,
    y = test_neg_binom_regression_data_not_linear$y
  )
our_implementation_neg_binom_not_linear
```

```
##                Estimate Std.Error     z.value      p.value DegOfFreedom
## (Intercept) -0.01177346 0.3735470 -0.03151802 1.0251435753           97
## x1           1.61130710 0.6804803  2.36789678 0.0178895261           NA
## x2           2.38360086 0.7053205  3.37945779 0.0007262896           NA
```

```
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] 10.62989
```

```
# 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.

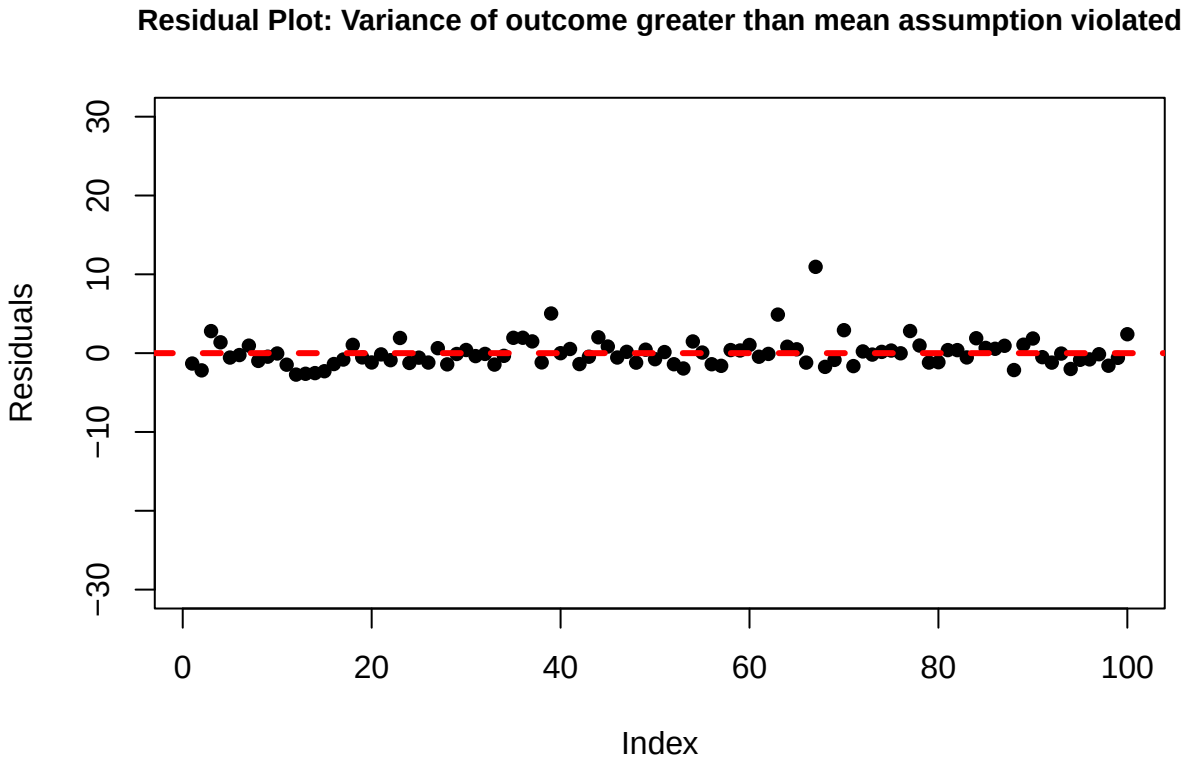
```
our_implementation_neg_binom_mean_greater <-
  negative_binomial_regression(
    test_neg_binom_regression_data_mean_greater,
    test_neg_binom_regression_data_mean_greater$x1,
    test_neg_binom_regression_data_mean_greater$x2,
    y = test_neg_binom_regression_data_mean_greater$y
  )
our_implementation_neg_binom_mean_greater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

```
##               Estimate  Std.Error    z.value    p.value DegOfFreedom
## (Intercept)  2.4669121 0.02186545 112.822403 0.00000000          97
## x1          -0.2112038 0.03748703  -5.634050 1.99999998          NA
## x2           0.1356869 0.04425716   3.065874 0.00217035          NA
```

```r
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] 1.813516
```

```r
# 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
)
```



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                 1.962647
## Linearity assumption violated      10.629895
## Variance > Mean assumption violated  1.813516
```

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.

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.