Homework - 3

Group 04

Problem 1: Gradient Descent Algorithm for Multiple Linear Regression

The file concrete.csv includes 1,030 types of concrete with numerical features indicating characteristics of the concrete. The variable "strength" is treated as the response variable.

- Standardize all variables (including the response variable "strength"). Split the data set into a training set (60%) and a validation set (40%).
- Implement the gradient descent algorithm in R with the ordinary least square cost function.
- Fit the multiple linear regression model using the gradient descent algorithm and the training set. Try out different learning rates: alpha = 0.01,0.1,0.3,0.5 and compare the speed of convergence by plotting the cost function. Determine the number of iterations needed for each alpha value.
- Apply the fitted regression model to the validation set and evaluate the model performance (ME, RMSE, MAE, MPE, MPAE). Calculate the correlation between the predicted strength and the actual strength. Create a lift chart to show model performance.

```r
# Import Required Packages
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```r
library(reshape2)
```

```
##
## Attaching package: 'reshape2'

## The following objects are masked from 'package:data.table':
##
```

```
##     dcast, melt
```

```r
library(MLmetrics)
```

```
##
## Attaching package: 'MLmetrics'

## The following object is masked from 'package:base':
##
##     Recall
```

```r
library(ggplot2)

# Read the csv file
df <- data.table(read.csv("concrete.csv"))

# Scale the dataframe
df <- as.data.frame(scale(df))

# Split into train and validation datasets
training_rows <- sample(seq_len(nrow(df)), size = floor(0.6 * nrow(df)))

train_data <- df[training_rows, ]
validation_data <- df[-training_rows, ]
```

Implementing Gradient Descent algorithm with the Ordinary Least Square cost function.

```r
# Define the gradient descent function
gradient_desc <- function(x, y, lr, iters) {
  # First we create a list to keep the track
  # of the cost function for each iteration
  losses <- list()

  # Convert y to a matrix
  y <- as.matrix(y)

  # create a column of 1
  ones <- rep(1, dim(x)[[1]])
  # append it to the input (this is our X0)
  X <- as.matrix(cbind(ones, x))
  # Calculate number of samples
  n <- length(y)

  # Initialize model parameters/coefficients
  theta <- as.matrix(rnorm(n = dim(X)[2], 0, 1))

  # Calculate model predictions
  y_hat <- X %*% theta

  # calculate the loss using OLS cost function
  loss <- sum((y_hat - y)^2) / (2 * n)

  # Calculate the gradients of the cost function
  grads <- t(X) %*% (y_hat - y)

  # Update theta
```

```r
  theta <- theta - lr * (1 / n) * grads


  # That was the first iteration of the gradient descent algorithm
  # Let's add the cost function to the list
  losses[[1]] <- loss


  counter <- 0
  # Number of iterations required to get the lowest loss
  sufficient_iterations <- 0
  for (i in 1:iters) {
    # Calculate model predictions
    y_hat <- X %*% theta

    # Calculate the loss using OLS cost function
    loss <- sum((y_hat - y)^2) / (2 * n)

    # Calculate the gradients
    grads <- t(X) %*% (y_hat - y)

    # Update theta
    theta <- theta - lr * (1 / n) * grads

    # Add cost to the list
    losses[[i + 1]] <- loss

    if (round(losses[[i]], 4) <= round(loss, 4)) {
      if (counter > 6) {
        break
      } else {
        counter <- counter + 1
        sufficient_iterations <- sufficient_iterations + 1
      }
    } else {
      counter <- 0
      sufficient_iterations <- sufficient_iterations + 1
    }
  }


  sufficient_iterations <- sufficient_iterations - counter
  # return the theta (aka model weights)
  return(list(
    "coeffs" = theta,
    "losses" = losses,
    "iterations_required" = sufficient_iterations,
    "final_loss" = loss
  ))
}

# Predict function
predict <- function(x, theta) {
```

```r
  ones <- rep(1, dim(x)[[1]])
  # append it to the input (this is our X0)
  X <- as.matrix(cbind(ones, x))

  return(X %*% t(theta))
}
```

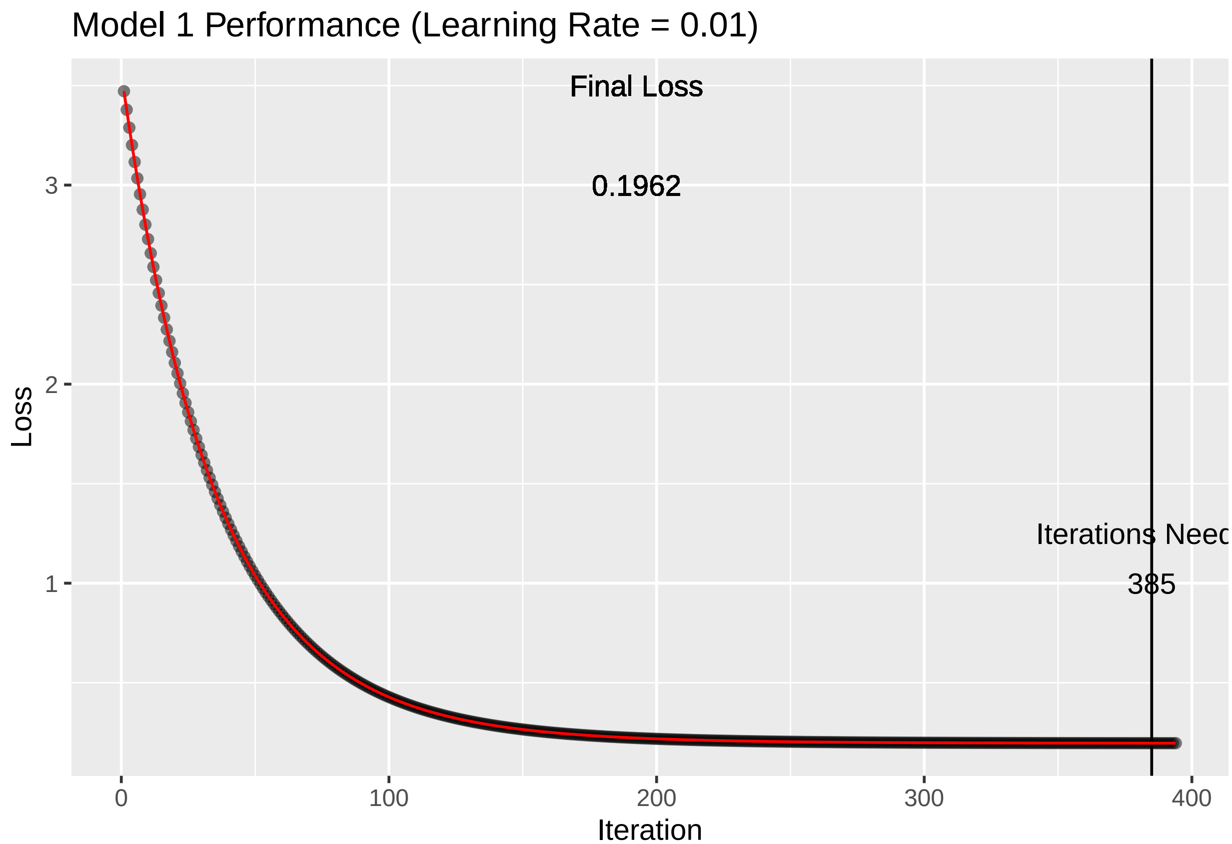
Now we create and train 4 models each with a different learning rate

```r
# Model 1, lr = 0.01
model1 <- gradient_desc(train_data[, 1:8], train_data$strength, lr = 0.01, iters = 10000)

model1_weights <- t(model1$coeffs)
model1_losses <- melt(data.frame(model1$losses))
model1_losses$index <- 1:dim(model1_losses)[[1]]



# Model 2, lr = 0.10
model2 <- gradient_desc(train_data[, 1:8], train_data$strength, lr = 0.10, iters = 10000)

model2_weights <- t(model2$coeffs)
model2_losses <- melt(data.frame(model2$losses))
model2_losses$index <- 1:dim(model2_losses)[[1]]

# Model 3, lr = 0.30
model3 <- gradient_desc(train_data[, 1:8], train_data$strength, lr = 0.30, iters = 10000)

model3_weights <- t(model3$coeffs)
model3_losses <- melt(data.frame(model3$losses))
model3_losses$index <- 1:dim(model3_losses)[[1]]

# Model 4, lr = 0.50
model4 <- gradient_desc(train_data[, 1:8], train_data$strength, lr = 0.50, iters = 10000)

model4_weights <- t(model4$coeffs)
model4_losses <- melt(data.frame(model4$losses))
model4_losses$index <- 1:dim(model4_losses)[[1]]
```

Let's plot the loss vs number of iterations for each model to evaluate their performance.

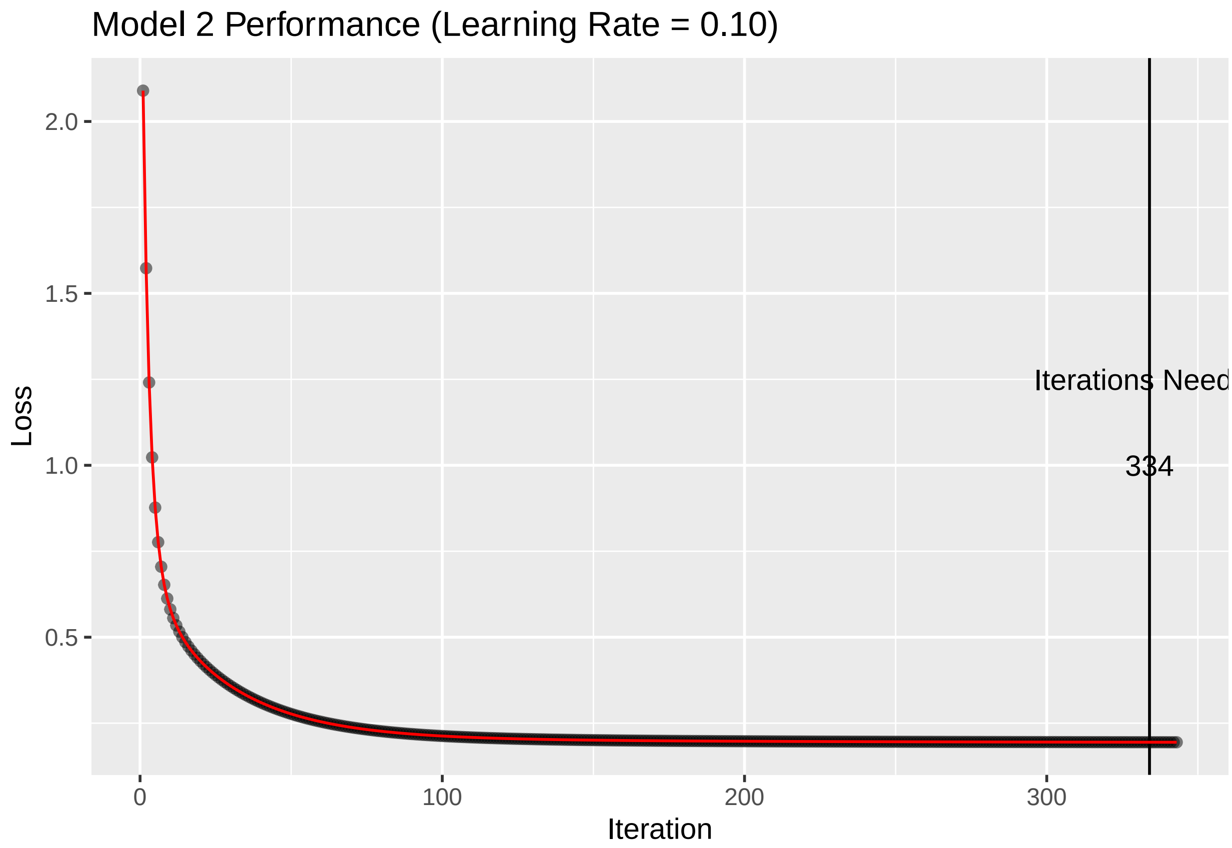
```r
# Model 1
ggplot(model1_losses, aes(x = index, y = value)) +
  geom_point(alpha = 0.5) +
  geom_vline(xintercept = model1$iterations_required) +
  geom_text(x = model1$iterations_required / 2, y = 3.5, label = "Final Loss") +
  geom_text(x = model1$iterations_required / 2, y = 3, label = as.character(round(model1$final_loss, 4))
  geom_text(
    x = model1$iterations_required,
    y = 1,
    label = as.character(model1$iterations_required),
    check_overlap = TRUE
  ) +
  geom_text(
    x = model1$iterations_required,
    y = 1.25,
```

```
    label = "Iterations Needed",
    check_overlap = TRUE
  ) +
  geom_line(color = "red") +
  labs(x = "Iteration", y = "Loss") +
  ggtitle("Model 1 Performance (Learning Rate = 0.01)")
```

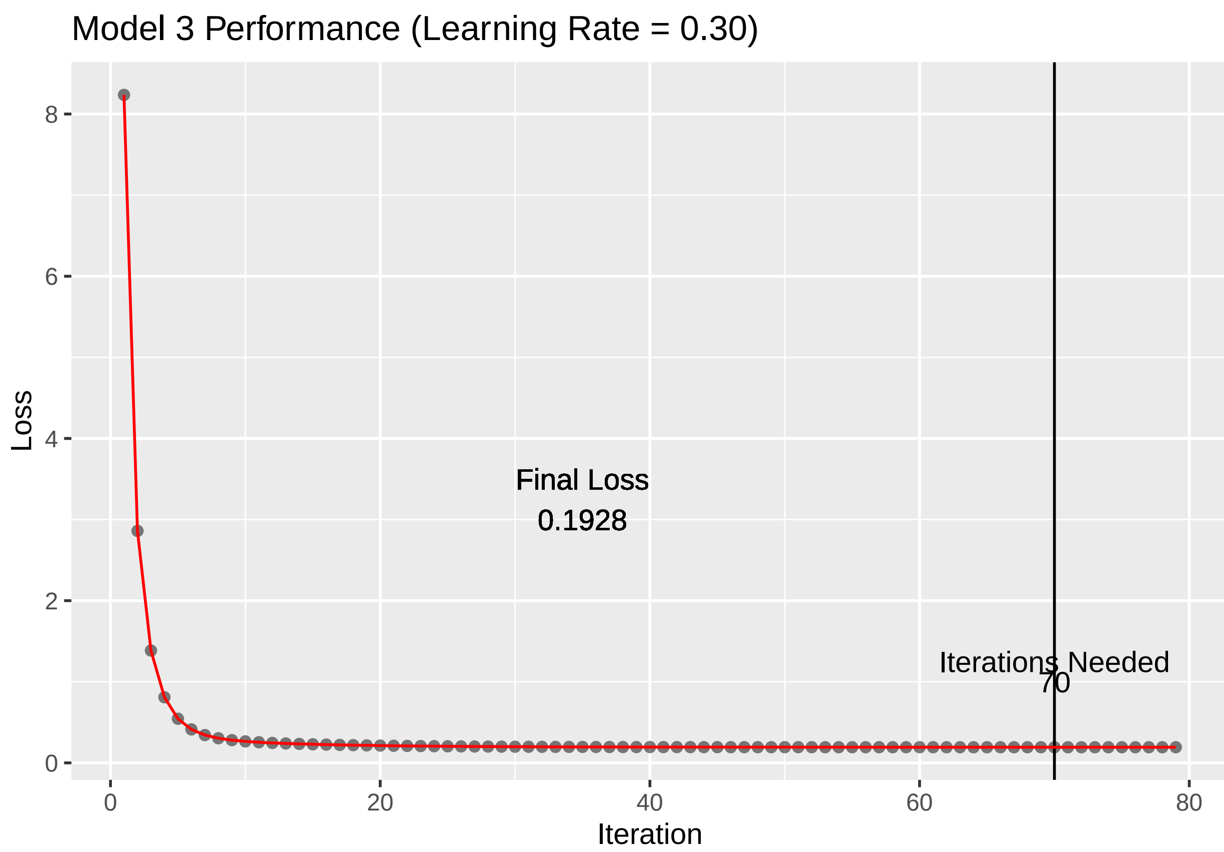



```
# Model 2
ggplot(model2_losses, aes(x = index, y = value)) +
  geom_point(alpha = 0.5) +
  geom_vline(xintercept = model2$iterations_required) +
  geom_text(x = model2$iterations_required / 2, y = 3.5, label = "Final Loss") +
  geom_text(x = model2$iterations_required / 2, y = 3, label = as.character(round(model2$final_loss, 4))
  geom_text(
    x = model2$iterations_required,
    y = 1,
    label = as.character(model2$iterations_required),
    check_overlap = TRUE
  ) +
  geom_text(
    x = model2$iterations_required,
    y = 1.25,
    label = "Iterations Needed",
    check_overlap = TRUE
  ) +
  geom_line(color = "red") +
```

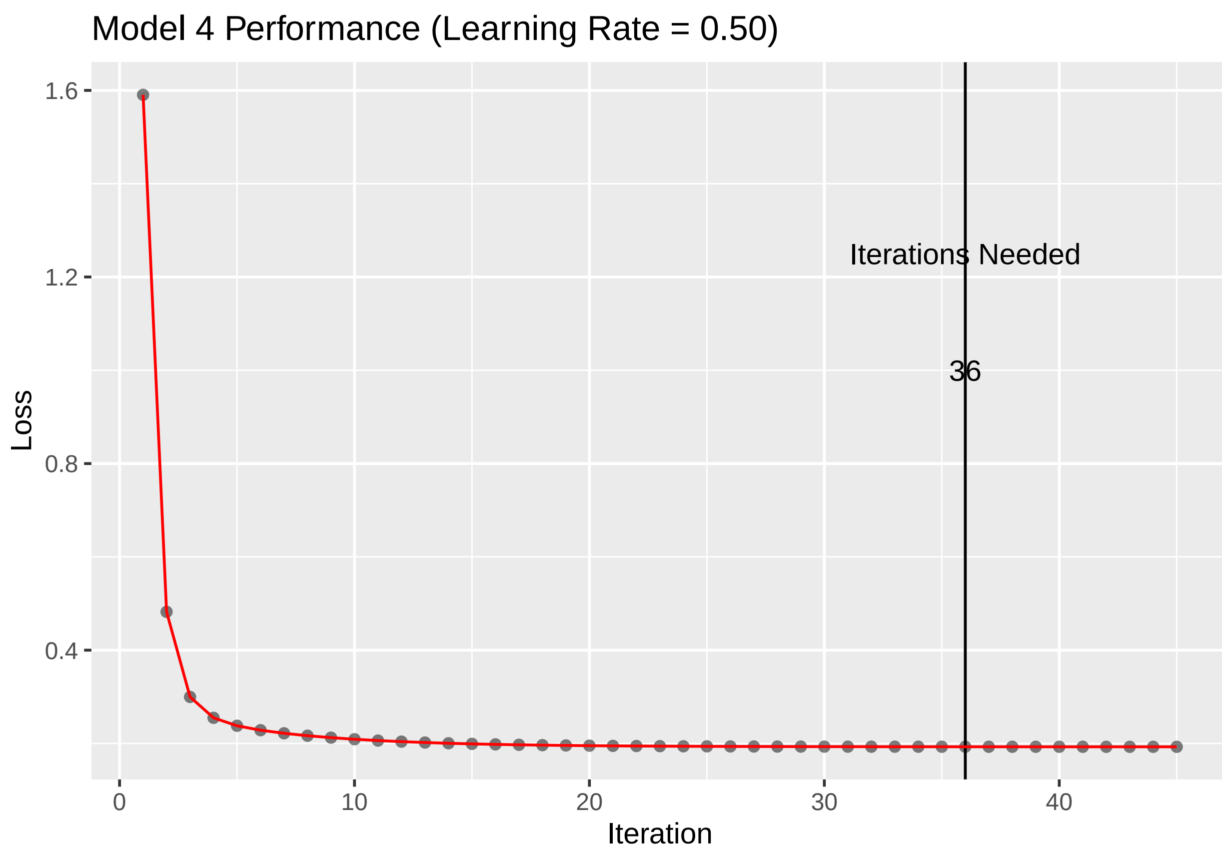
```
  labs(x = "Iteration", y = "Loss") +
  ggtitle("Model 2 Performance (Learning Rate = 0.10)")
```




```
# Model 3
ggplot(model3_losses, aes(x = index, y = value)) +
  geom_point(alpha = 0.5) +
  geom_vline(xintercept = model3$iterations_required) +
  geom_text(x = model3$iterations_required / 2, y = 3.5, label = "Final Loss") +
  geom_text(x = model3$iterations_required / 2, y = 3, label = as.character(round(model3$final_loss, 4))
  geom_text(
    x = model3$iterations_required,
    y = 1,
    label = as.character(model3$iterations_required),
    check_overlap = TRUE
  ) +
  geom_text(
    x = model3$iterations_required,
    y = 1.25,
    label = "Iterations Needed",
    check_overlap = TRUE
  ) +
  geom_line(color = "red") +
  labs(x = "Iteration", y = "Loss") +
  ggtitle("Model 3 Performance (Learning Rate = 0.30)")
```




```r
# Model 4
ggplot(model4_losses, aes(x = index, y = value)) +
  geom_point(alpha = 0.5) +
  geom_vline(xintercept = model4$iterations_required) +
  geom_text(x = model4$iterations_required / 2, y = 3.5, label = "Final Loss") +
  geom_text(x = model4$iterations_required / 2, y = 3, label = as.character(round(model4$final_loss, 4)
  geom_text(
    x = model4$iterations_required,
    y = 1,
    label = as.character(model4$iterations_required),
    check_overlap = TRUE
  ) +
  geom_text(
    x = model4$iterations_required,
    y = 1.25,
    label = "Iterations Needed",
    check_overlap = TRUE
  ) +
  geom_line(color = "red") +
  labs(x = "Iteration", y = "Loss") +
  ggtitle("Model 4 Performance (Learning Rate = 0.50)")
```




```
cat("Number of iterations required for each model are :\n")
```

```
## Number of iterations required for each model are :
```

```
cat("Model 1:", as.character(model1$iterations_required), "\n")
```

```
## Model 1: 385
```

```
cat("Model 2:", as.character(model2$iterations_required), "\n")
```

```
## Model 2: 334
```

```
cat("Model 3:", as.character(model3$iterations_required), "\n")
```

```
## Model 3: 70
```

```
cat("Model 4:", as.character(model4$iterations_required), "\n")
```

```
## Model 4: 36
```

As observed, the model converges faster as the learning rate increases.

Testing the model on the validation data and calulcating errors -

```
# We define the Mean Error function
ME <- function(y_hat, y) {
  sum(y - y_hat) / length(y)
}

# We define the Mean Percentage Error Function
MPE <- function(y_hat, y) {
```

```r
  (sum((y - y_hat) / y)) / length(y)
}
```

Now let's look at the model statstics -

```r
model1_predictions <- predict(validation_data[, 1:8], model1_weights)

cat("----Model 1 Summary ----\n")
```

```
## ----Model 1 Summary ----
```

```r
cat("MAE:", MAE(model1_predictions, validation_data[, 9]), "\n")
```

```
## MAE: 0.4862348
```

```r
cat("RMSE:", RMSE(model1_predictions, validation_data[, 9]), "\n")
```

```
## RMSE: 0.6193269
```

```r
cat("ME:", ME(model1_predictions, validation_data[, 9]), "\n")
```

```
## ME: -0.03444517
```

```r
cat("MPE:", MPE(model1_predictions, validation_data[, 9]), "\n")
```

```
## MPE: 0.4457131
```

```r
cat("MPAE", MAPE(model1_predictions, validation_data[, 9]), "\n")
```

```
## MPAE 1.813482
```

```r
model2_predictions <- predict(validation_data[, 1:8], model2_weights)

cat("----Model 2 Summary ---- \n")
```

```
## ----Model 2 Summary ----
```

```r
cat("MAE:", MAE(model2_predictions, validation_data[, 9]), "\n")
```

```
## MAE: 0.498266
```

```r
cat("RMSE:", RMSE(model2_predictions, validation_data[, 9]), "\n")
```

```
## RMSE: 0.6270598
```

```r
cat("ME:", ME(model2_predictions, validation_data[, 9]), "\n")
```

```
## ME: -0.01841473
```

```r
cat("MPE:", MPE(model2_predictions, validation_data[, 9]), "\n")
```

```
## MPE: 0.2837325
```

```r
cat("MPAE", MAPE(model2_predictions, validation_data[, 9]), "\n")
```

```
## MPAE 2.111422
```

```r
model3_predictions <- predict(validation_data[, 1:8], model3_weights)

cat("----Model 3 Summary ----\n")
```

```
## ----Model 3 Summary ----
```

```r
cat("MAE:", MAE(model3_predictions, validation_data[, 9]), "\n")
```

```
## MAE: 0.4914922
```

```r
cat("RMSE:", RMSE(model3_predictions, validation_data[, 9]), "\n")
```

```
## RMSE: 0.6208526
```

```r
cat("ME:", ME(model3_predictions, validation_data[, 9]), "\n")
```

```
## ME: -0.01422963
```

```r
cat("MPE:", MPE(model3_predictions, validation_data[, 9]), "\n")
```

```
## MPE: 0.3826415
```

```r
cat("MPAE", MAPE(model3_predictions, validation_data[, 9]), "\n")
```

```
## MPAE 2.022136
```

```r
model4_predictions <- predict(validation_data[, 1:8], model4_weights)

cat("----Model 4 Summary ----\n")
```

```
## ----Model 4 Summary ----
```

```r
cat("MAE:", MAE(model4_predictions, validation_data[, 9]), "\n")
```

```
## MAE: 0.4916012
```

```r
cat("RMSE:", RMSE(model4_predictions, validation_data[, 9]), "\n")
```

```
## RMSE: 0.6210676
```

```r
cat("ME:", ME(model4_predictions, validation_data[, 9]), "\n")
```

```
## ME: -0.01566395
```

```r
cat("MPE:", MPE(model4_predictions, validation_data[, 9]), "\n")
```

```
## MPE: 0.3318516
```

```r
cat("MPAE", MAPE(model4_predictions, validation_data[, 9]), "\n")
```

```
## MPAE 2.004415
```

We can see that all the models have approximately the same accuracy regardless the learning rate.

Calculating the correlation between predicted strength and actual strength

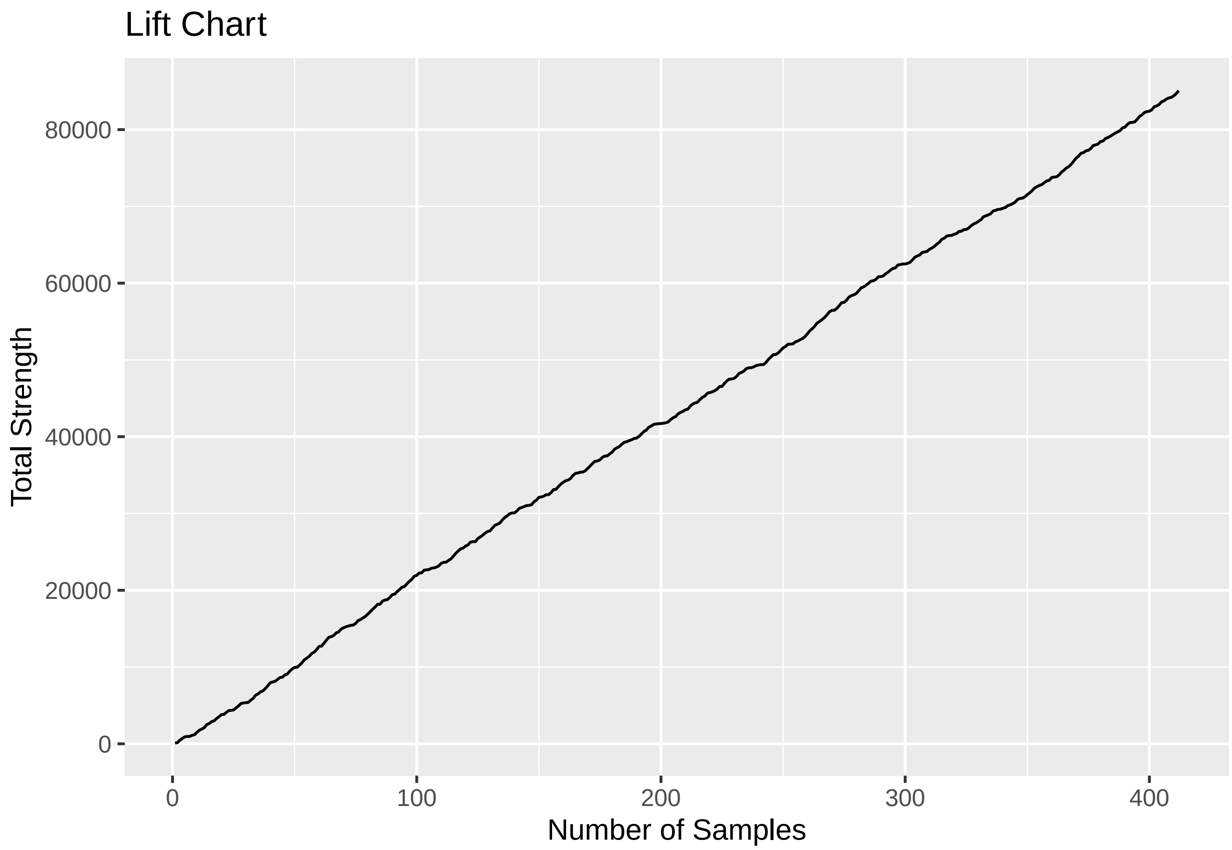
```r
cat("The correlation is :", cor(model1_predictions, validation_data[, 9]), "\n")
```

```
## The correlation is : 0.788954
```

Plotting a lift chart

```r
# Create a temp data frame to calculate the sumulative strength
temp <- data.frame("strength" = order(validation_data[, 9]))
temp$cumstrength <- cumsum(temp$strength)
temp$samples <- 1:dim(temp)[[1]]

# Plot the lift chart
ggplot(temp, aes(x = samples, y = cumstrength)) +
  geom_line() +
```

```r
  labs(x = "Number of Samples", y = "Total Strength") +
  ggtitle("Lift Chart")
```




```r
# Delete all environment variables
rm(list = ls())
```

Problem 2