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.

```
# 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
```

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

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

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

```
library(reshape2)
```

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

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

```
##     dcast, melt
```

```
library(car)
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
library(MLmetrics)
```

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

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

```
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.

```
# 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
```

```r
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
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) {
  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))
}
```

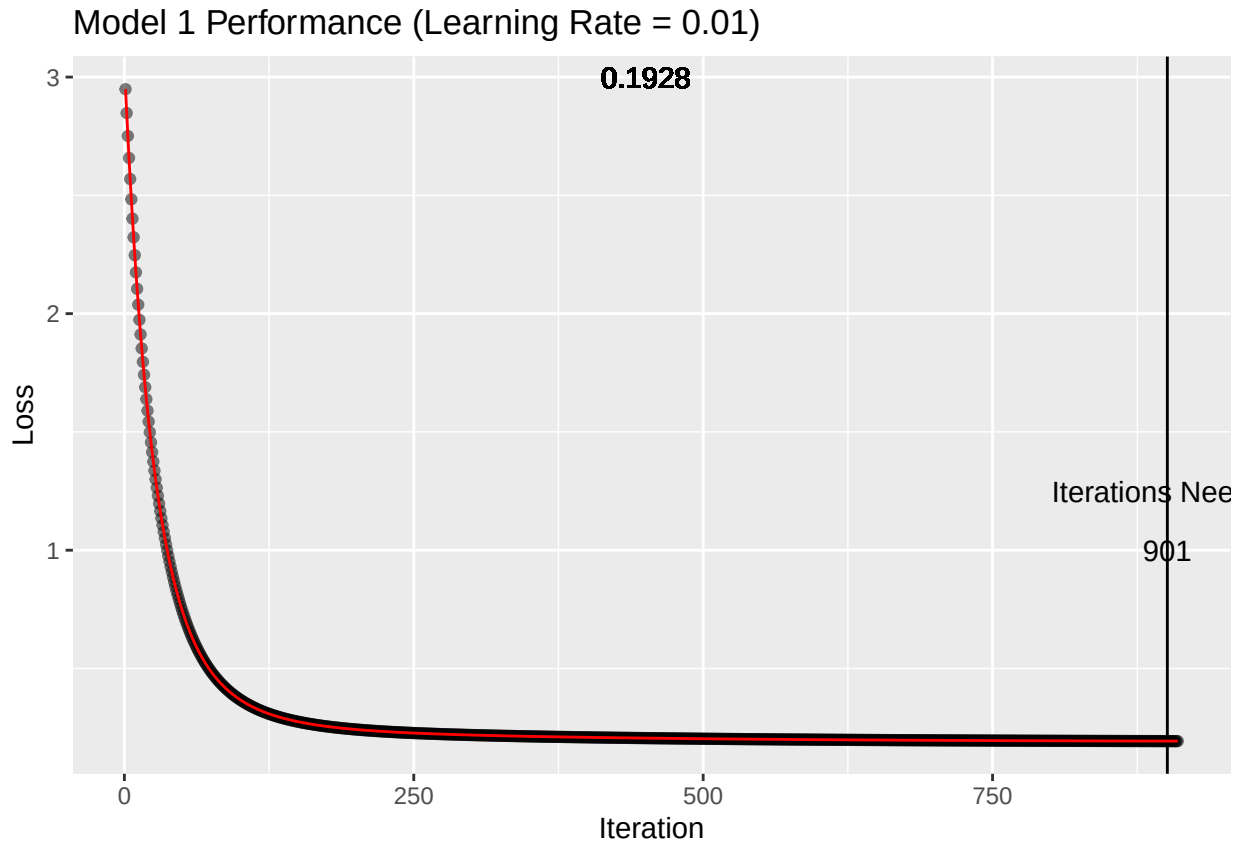
Now we create and train 4 models each with a different learning rate

```
# 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.

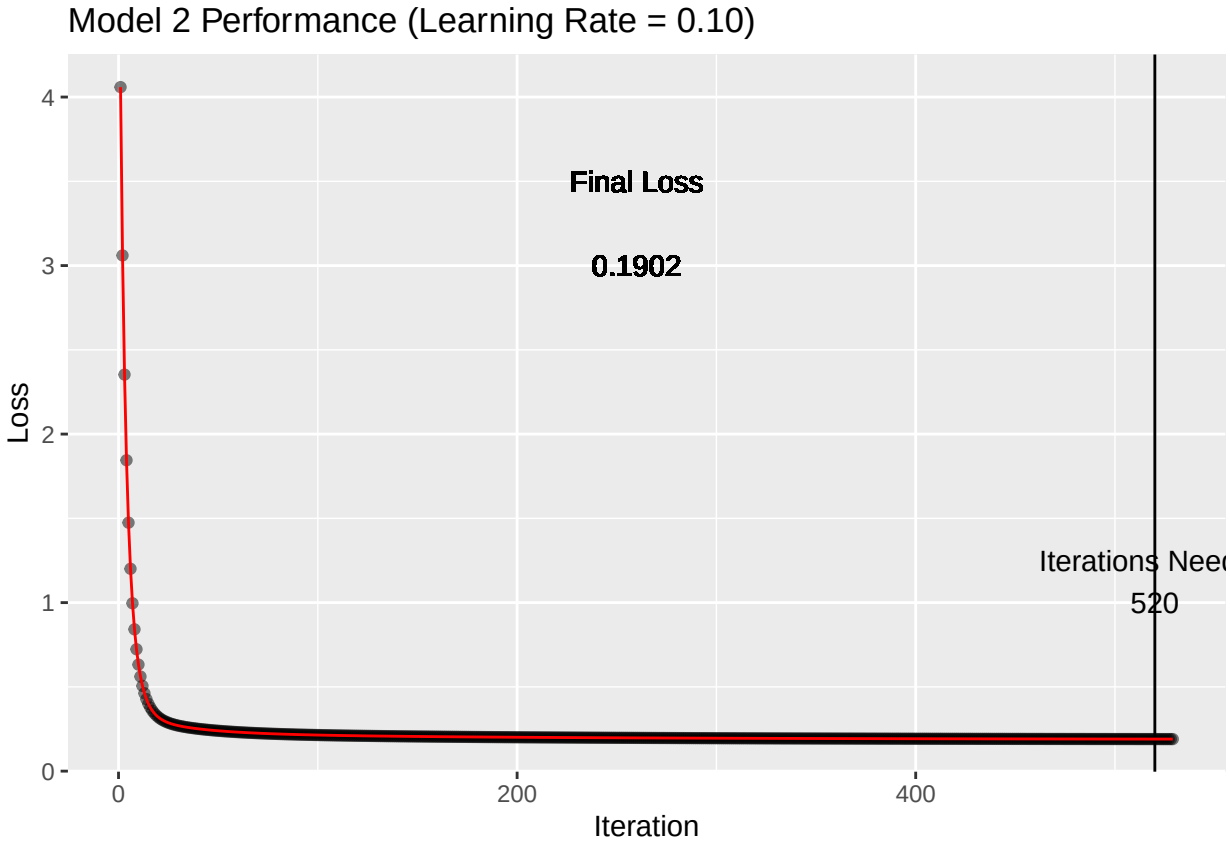
```
# Model 1
ggplot(model1_losses, aes(x = index, y = value)) +
  geom_point(alpha = 0.5) +
  geom_vline(xintercept = model1$iterations_required) +
  geom_text(x = model1$iterations_required / 2, y = 3.5, label = "Final Loss") +
  geom_text(x = model1$iterations_required / 2, y = 3, label = as.character(round(model1$final_loss,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



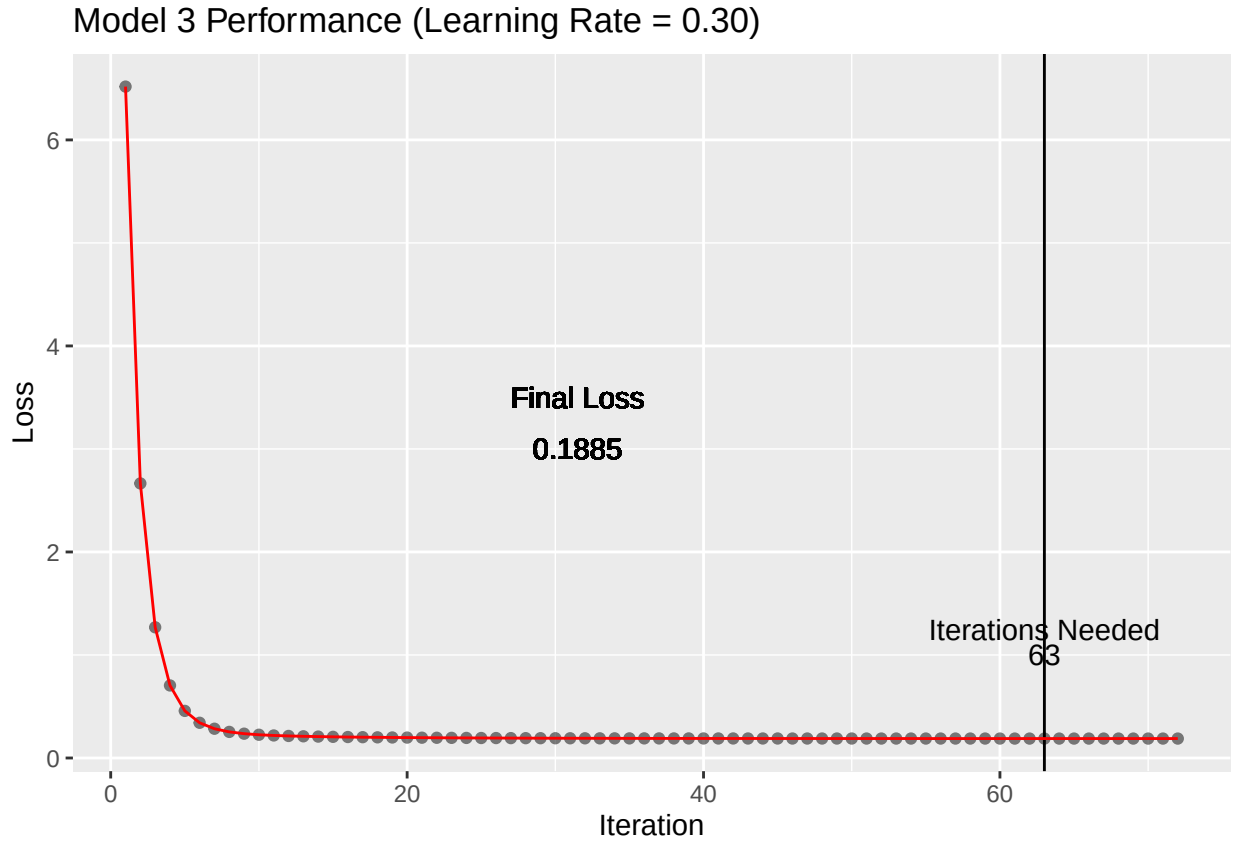

```
# Model 2
ggplot(model2_losses, aes(x = index, y = value)) +
  geom_point(alpha = 0.5) +
  geom_vline(xintercept = model2$iterations_required) +
  geom_text(x = model2$iterations_required / 2, y = 3.5, label = "Final Loss") +
  geom_text(x = model2$iterations_required / 2, y = 3, label = as.character(round(model2$final_loss, 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)")
```



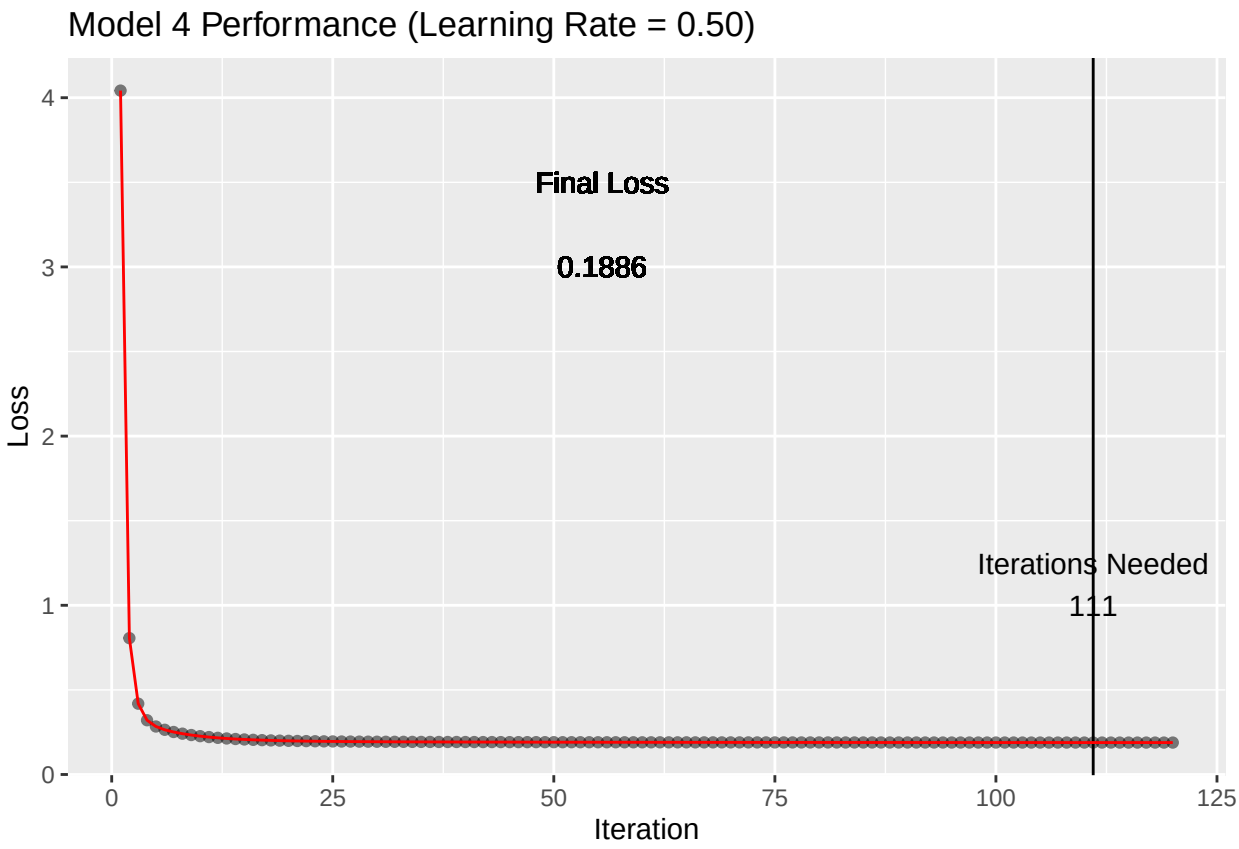

```
# Model 3
ggplot(model3_losses, aes(x = index, y = value)) +
  geom_point(alpha = 0.5) +
  geom_vline(xintercept = model3$iterations_required) +
  geom_text(x = model3$iterations_required / 2, y = 3.5, label = "Final Loss") +
  geom_text(x = model3$iterations_required / 2, y = 3, label = as.character(round(model3$final_loss, 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)")
```




```
# 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: 901
```

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

```
## Model 2: 520
```

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

```
## Model 3: 63
```

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

```
## Model 4: 111
```

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

Now let's look at the model statstics -

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

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

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

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

```
## MAE: 0.5030316
```

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

```
## RMSE: 0.639261
```

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

```
## ME: -0.05153276
```

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

```
## MPE: 0.170357
```

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

```
## MPAE 1.736467
```

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

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

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

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

```
## MAE: 0.5017961
```

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

```
## RMSE: 0.6388807
```

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

```
## ME: -0.05355958
```

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

```
## MPE: 0.1527597
```

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

```
## MPAE 1.777269
```

```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: 0.498239
```

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

```
## RMSE: 0.6350463
```

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

```
## ME: -0.04721611
```

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

```
## MPE: 0.2311124
```

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

```
## MPAE 1.704141
```

```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: 0.498534
```

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

```
## RMSE: 0.6350665
```

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

```
## ME: -0.04779882
```

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

```
## MPE: 0.2231632
```

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

```
## MPAE 1.709168
```

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

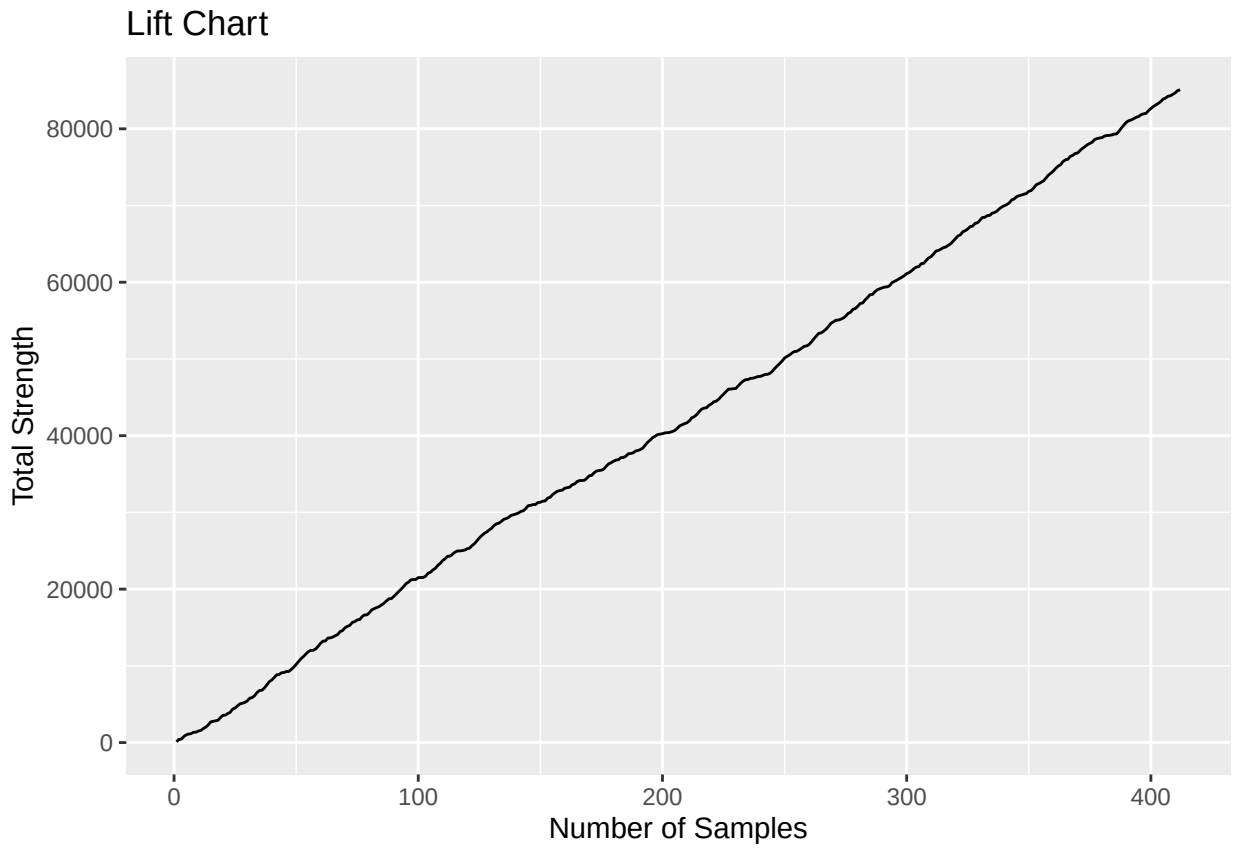
```r
cat("The correlation is :", cor(model1_predictions, validation_data[, 9]), "\n")
```

```
## The correlation is : 0.7567082
```

Plotting a lift chart

```r
# Create a temp data frame to calculate the sumulative strength
temp <- data.frame("strength" = order(validation_data[, 9]))
```

```r
temp$cumstrength <- cumsum(temp$strength)
temp$samples <- 1:dim(temp)[[1]]

# Plot the lift chart
ggplot(temp, aes(x = samples, y = cumstrength)) +
  geom_line() +
  labs(x = "Number of Samples", y = "Total Strength") +
  ggtitle("Lift Chart")
```




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

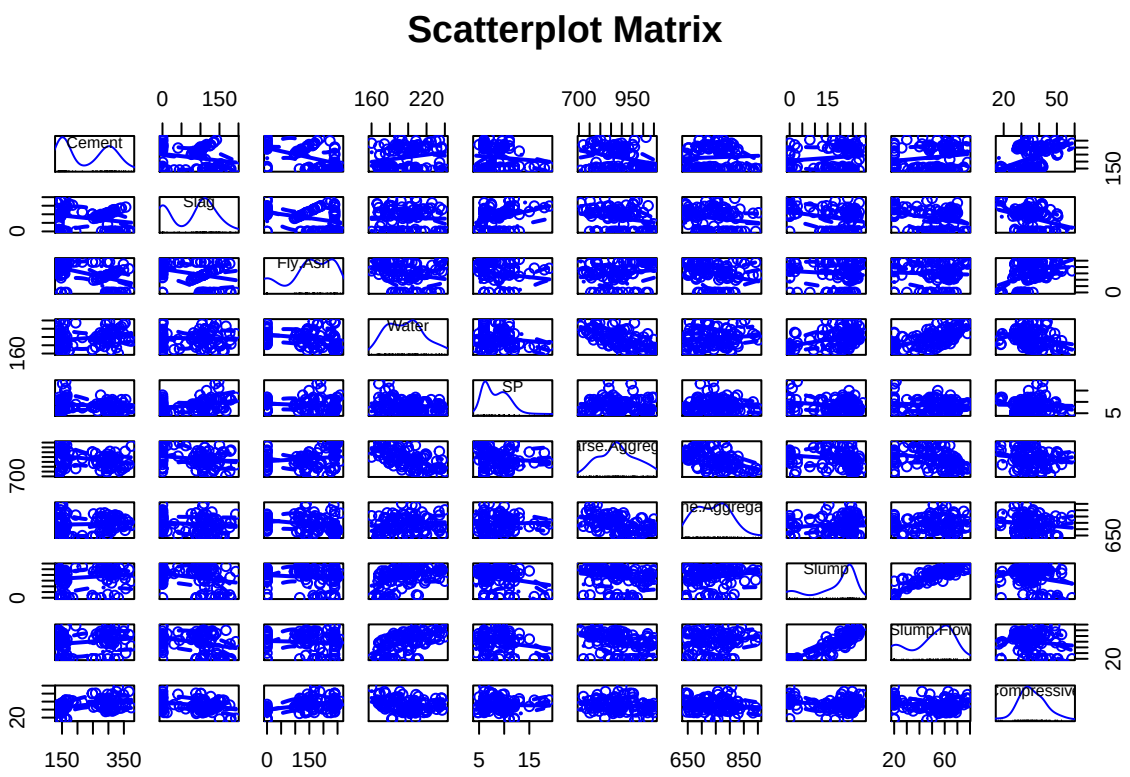
Problem 2

- Read the included research article "Modeling Slump Flow Concrete". It is sufficient to consider "Slump Flow" as the response variable in this problem just as in the included article.
- Create a scatterplot matrix of "Concrete Slump Test Data" and select an initial set of predictor variables.
- Build a few potential regression models using "Concrete Slump Test Data"
- Perform regression diagnostics using both typical approach and enhanced approach
- Identify unusual observations and take corrective measures
- Select the best regression model

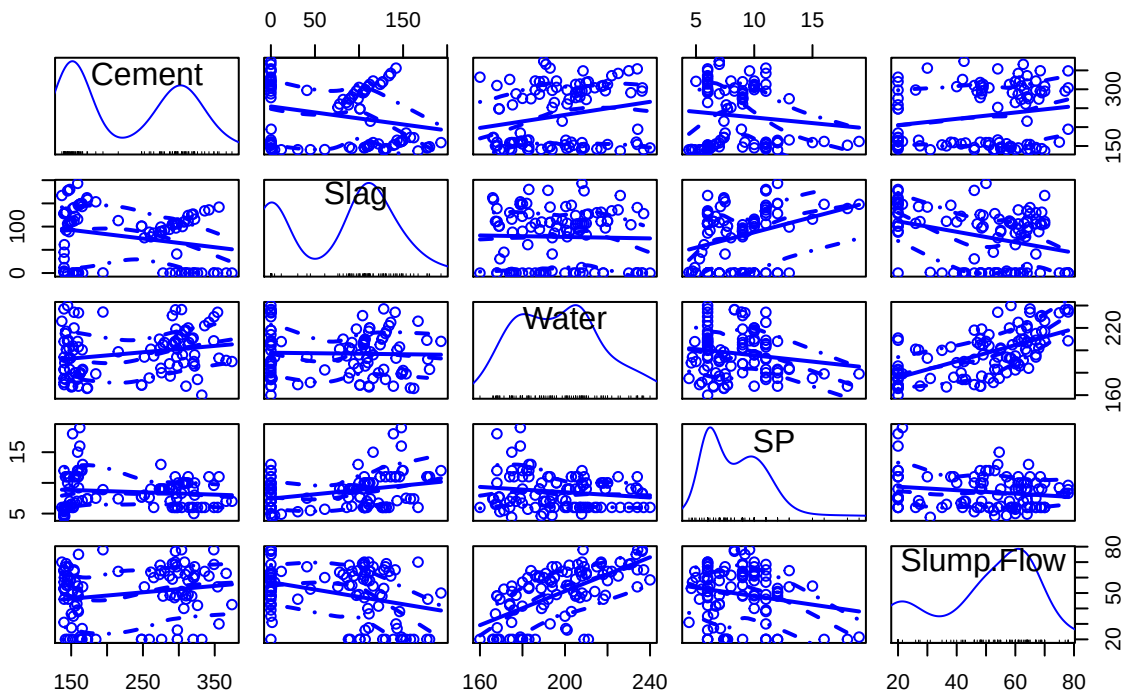
- Fine tune the selection of predictor variables
- Interpret the prediction results

In our opinion, it is sufficient to consider "Slump Flow" as the response variable because the slump flow is a function of the content of all concrete ingredients including cement, fly ash, blast furnace slag, water, superplasticizer, coarse aggregate, and fine aggregate. And since HPC is already so complicated to model, incorporating another response variable may overcomplicate the model hypothesis.

```
df <- readxl::read_xlsx("Concrete Slump Test Data.xlsx", sheet = "Concrete slump")
df <- df[, 2:11]

# Let's plot the scatterplot matrix
scatterplotMatrix(df, main = "Scatterplot Matrix")
```




```
# Since the above matrix is hard to interpret, we only plot it for a select
# variables
scatterplotMatrix(~ Cement + Slag + Water + SP + `Slump Flow`,
  data = df,
  main = "Scatterplot Matrix"
)
```



Let's build a few regression models using these predictor variables

```
fit1 <- lm(`Slump Flow` ~ Water + `Coarse Aggregate` + `Fine Aggregate`, data = df)

summary(fit1)
```

```
##
## Call:
## lm(formula = `Slump Flow` ~ Water + `Coarse Aggregate` + `Fine Aggregate`,
##     data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -30.163  -8.837   1.799   9.869  24.383
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)        -162.70980   44.17173  -3.684 0.000375 ***
## Water                 0.64760    0.08495   7.623 1.53e-11 ***
## `Coarse Aggregate`    0.04545    0.02211   2.055 0.042476 *
## `Fine Aggregate`      0.06011    0.02480   2.424 0.017165 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.37 on 99 degrees of freedom
## Multiple R-squared:  0.4376, Adjusted R-squared:  0.4205
## F-statistic: 25.67 on 3 and 99 DF,  p-value: 2.28e-12
```

```r
fit2 <- lm(`Slump Flow` ~ Water + Slag + `Fine Aggregate`, data = df)

summary(fit2)
```

```
##
## Call:
## lm(formula = `Slump Flow` ~ Water + Slag + `Fine Aggregate`,
##     data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -32.470 -10.428   2.035   9.123  22.867
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -62.61966   18.59310  -3.368  0.00108 **
## Water              0.53605    0.06221   8.617 1.12e-13 ***
## Slag              -0.08683    0.02101  -4.133 7.51e-05 ***
## `Fine Aggregate`   0.01799    0.02018   0.892  0.37477
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.61 on 99 degrees of freedom
## Multiple R-squared:  0.4998, Adjusted R-squared:  0.4847
## F-statistic: 32.98 on 3 and 99 DF,  p-value: 7.292e-15
```

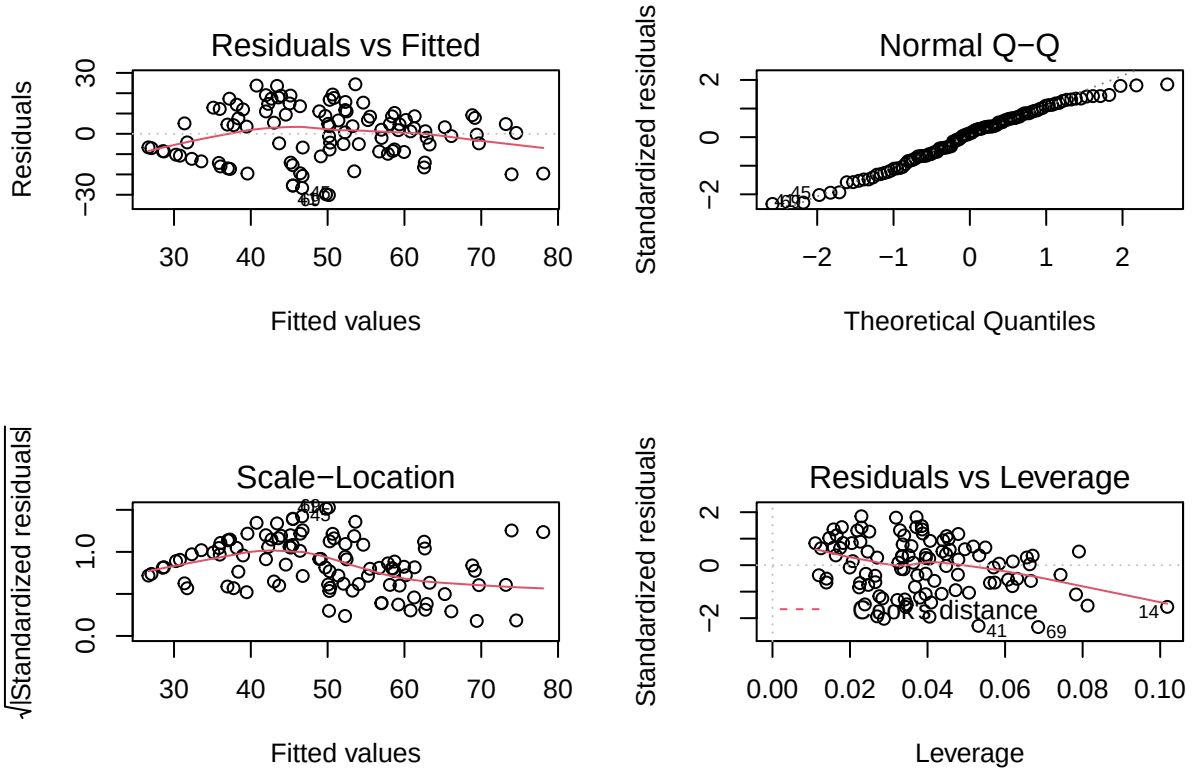
Let's try and fit a quadratic model

```r
fit3 <- lm(`Slump Flow` ~ (Water^2) + Water + Slag, data = df)

summary(fit3)
```

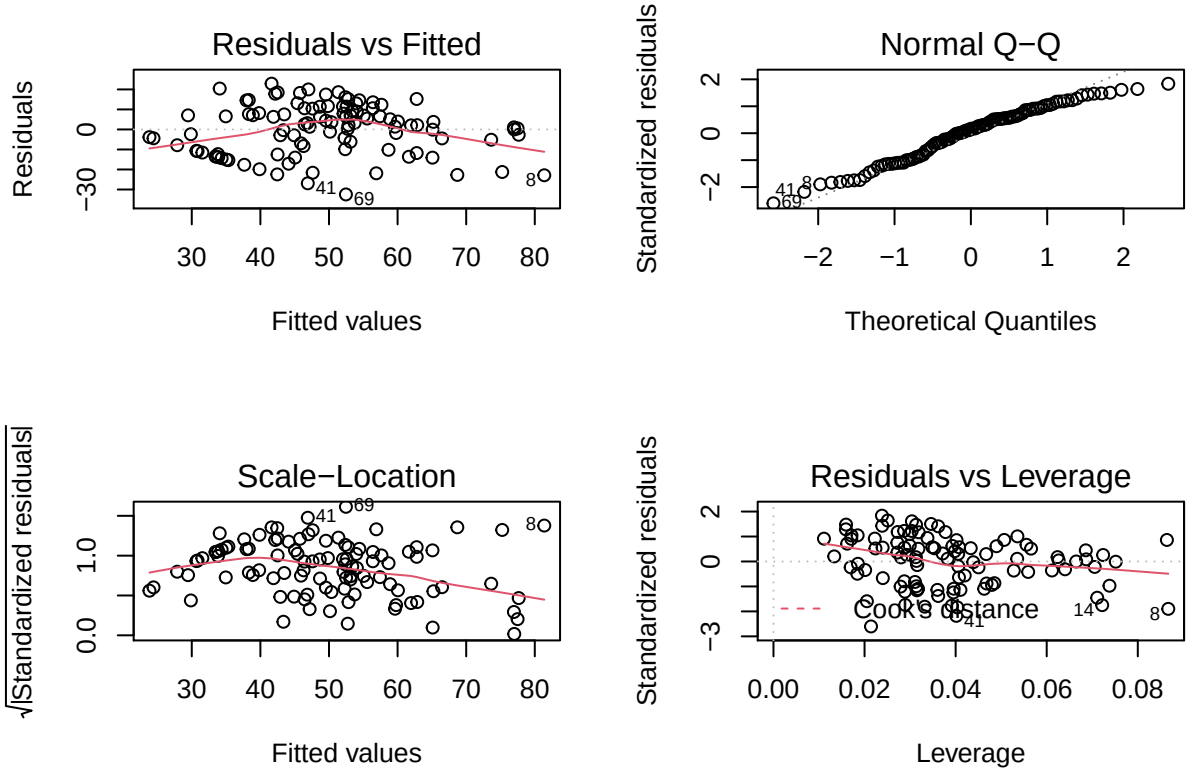
```
##
## Call:
## lm(formula = `Slump Flow` ~ (Water^2) + Water + Slag, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -32.687 -10.746   2.010   9.224  23.927
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -50.26656   12.38669  -4.058 9.83e-05 ***
## Water         0.54224    0.06175   8.781 4.62e-14 ***
## Slag         -0.09023    0.02064  -4.372 3.02e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.6 on 100 degrees of freedom
## Multiple R-squared:  0.4958, Adjusted R-squared:  0.4857
## F-statistic: 49.17 on 2 and 100 DF,  p-value: 1.347e-15
```

Performing Regression Diagnostics using Typical Approach

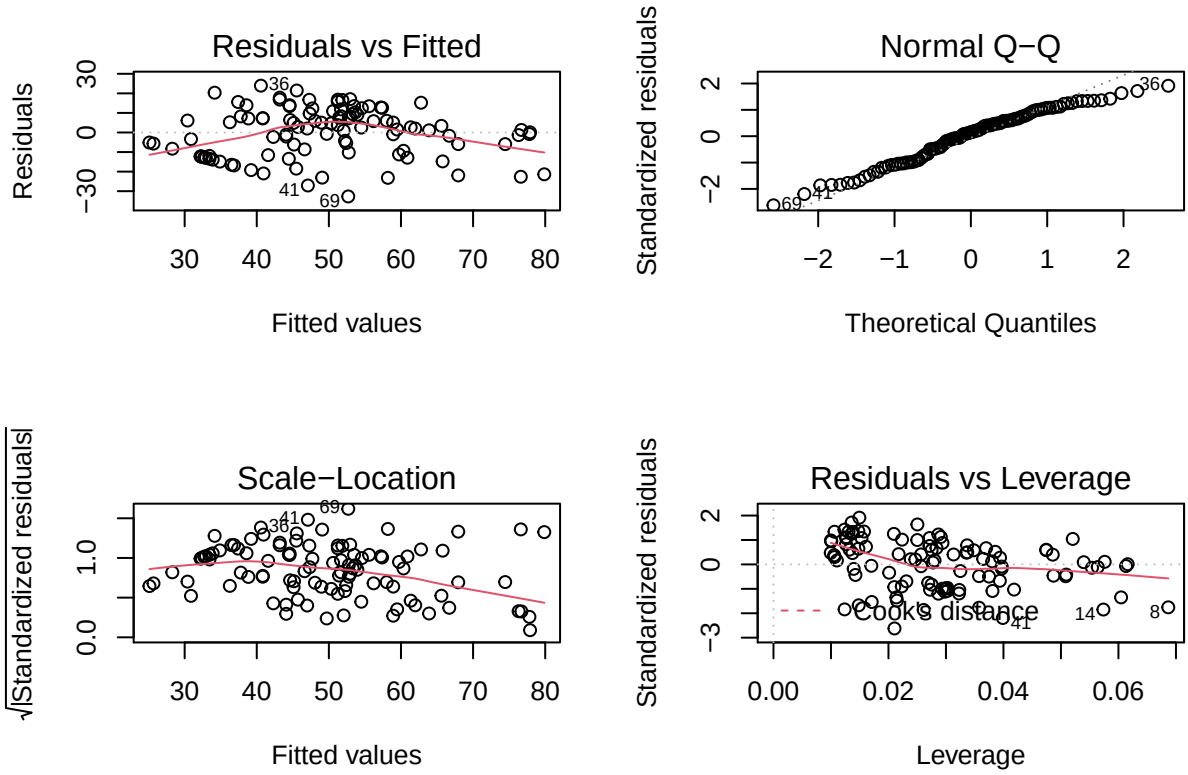
```
# Model 1
par(mfrow = c(2, 2))
plot(fit1)
```




```
# Model 2
par(mfrow = c(2, 2))
plot(fit2)
```

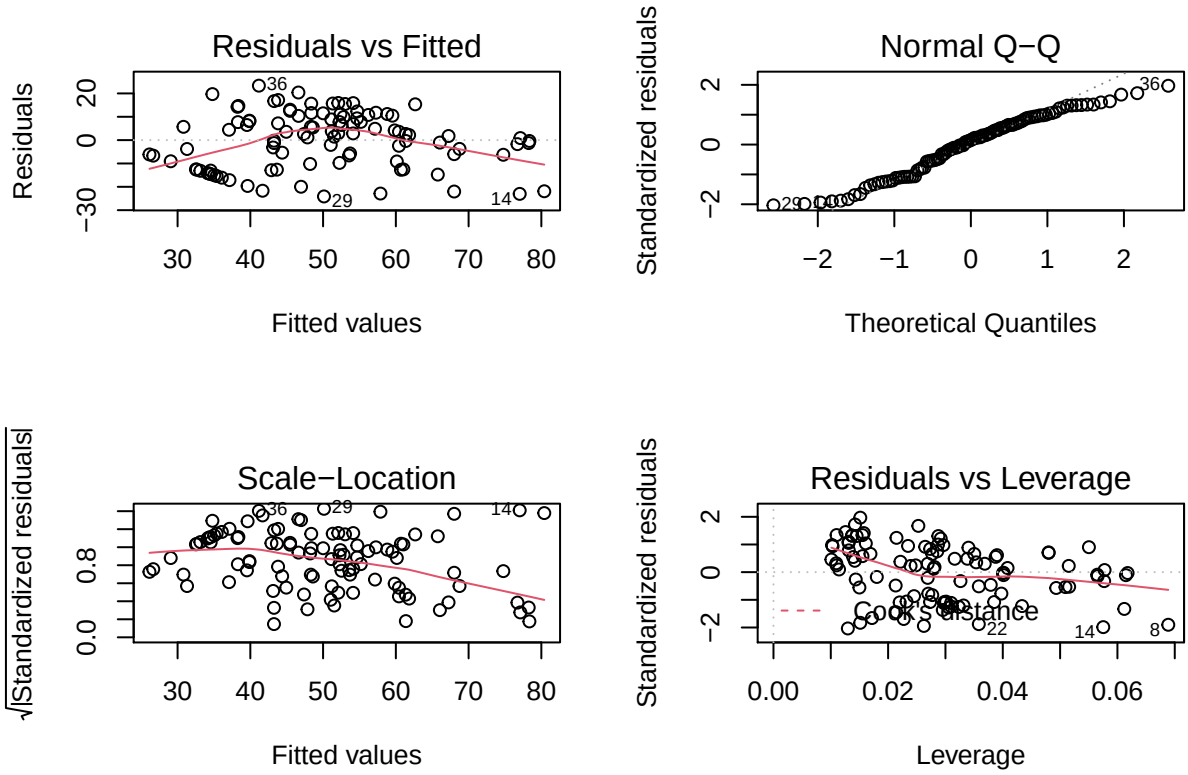



```
# Model 3
par(mfrow = c(2, 2))
plot(fit3)
```



Model 3 seems to be the best fit. We can also see that points 41 and 69 appear to be influential. We can remove these two points from the data to see if the model fits better.

```
fit3 <- lm(`Slump Flow` ~ (Water^2) + Water + Slag, data = df[-c(41, 69), ])

# Model 3
par(mfrow = c(2, 2))
plot(fit3)
```

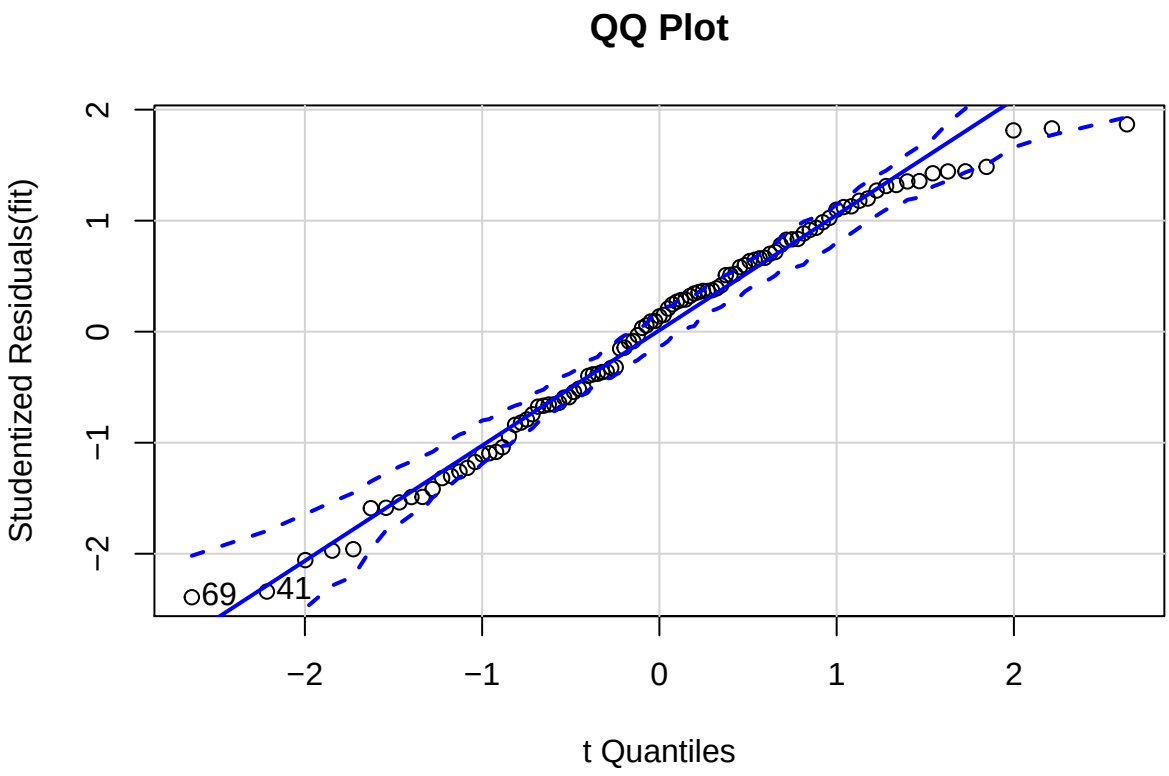


The model seems to fit the data quite well.

Peforming Diagnostic Regression with Enhanced Approach

Normality

```
fit <- lm(`Slump Flow` ~ Water + `Coarse Aggregate` + `Fine Aggregate`, data = df)

qqPlot(fit, labels = rownames(df), id.method = "identify", simulate = TRUE, main = "QQ Plot")
```




```
## [1] 41 69
```

We can see from the QQ Plot that our model satisfies normality. Almost all the points fall on the 45 degree line except for a few.

Independence of Errors

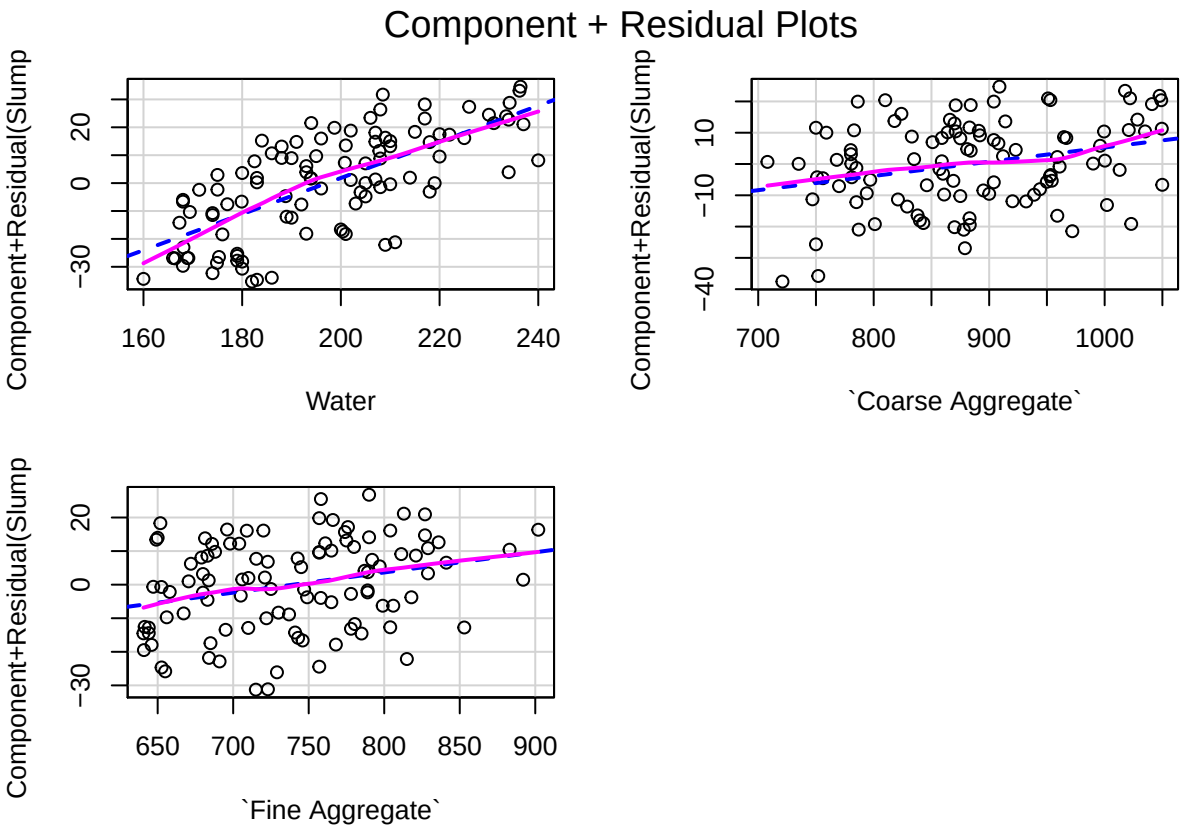
```
durbinWatsonTest(fit)
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1      0.06668866      1.830473   0.362
##  Alternative hypothesis: rho != 0
```

Since the p-value is insignificant, there is no autocorrelation and hence and independence of errors.

Linearity

```
crPlots(fit)
```



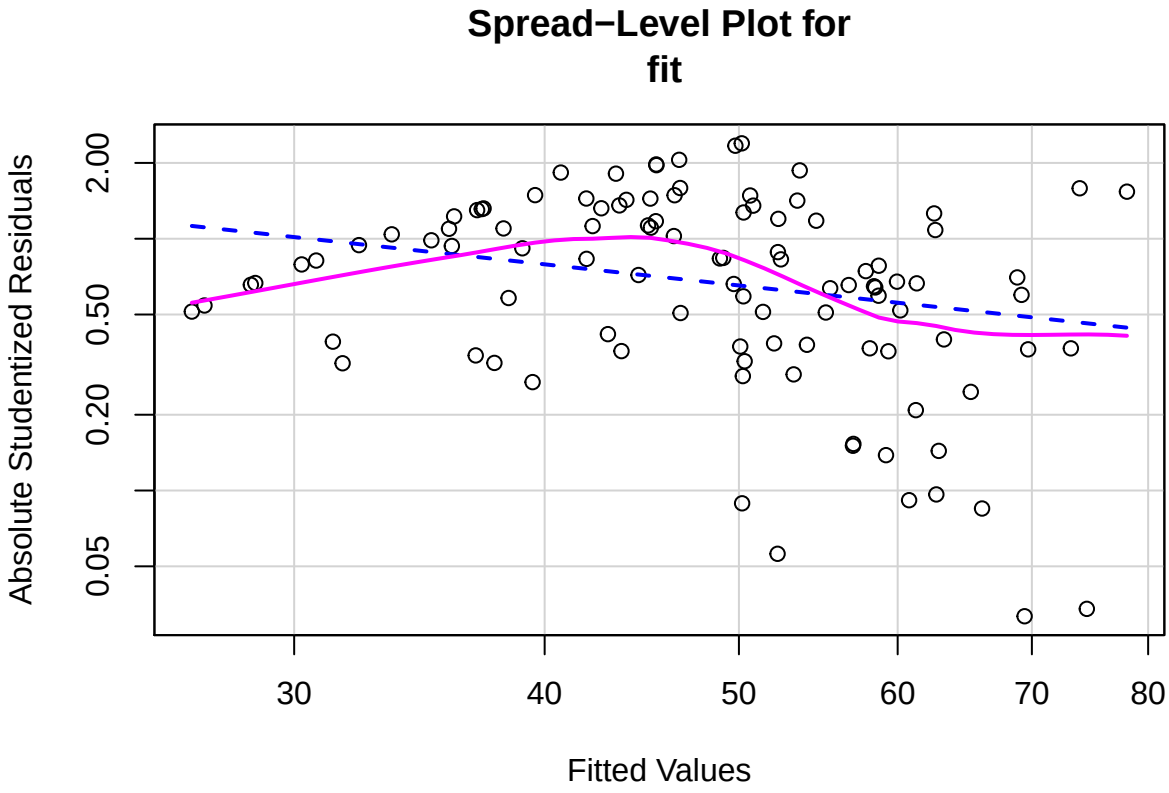
It seems that this model satisfies linearity.

Homoscedasticity

```
ncvTest(fit)
```

```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 1.533171, Df = 1, p = 0.21564
```

```
spreadLevelPlot(fit)
```




```
##
## Suggested power transformation:  1.866028
```

From the insignificant p-value and the Spread-Level Plot we can see that the model meets the requirements for Homoscedasticity.

Unusual Observations and Corrective Measures

```
outlierTest(fit)
```

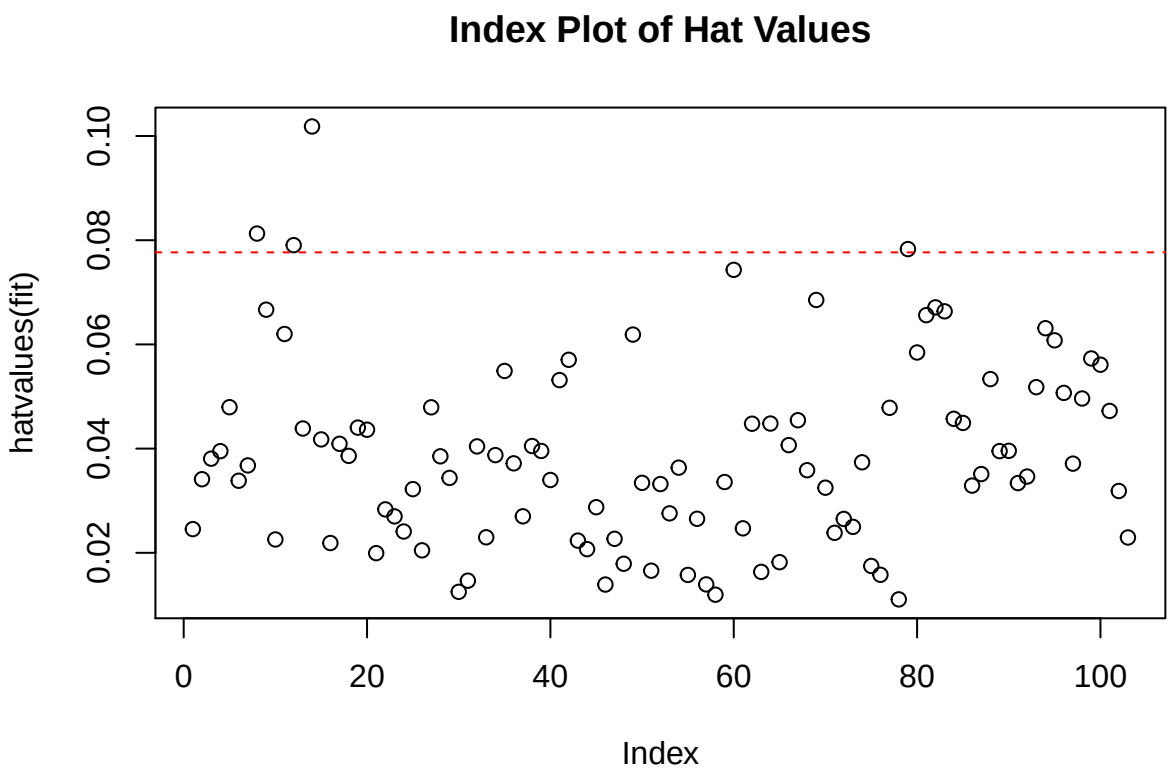
```
## No Studentized residuals with Bonferroni p < 0.05
## Largest |rstudent|:
##     rstudent unadjusted p-value Bonferroni p
## 69 -2.391905           0.018669           NA
```

We can see that point 69 is an outlier. But the p-value is not significant and hence we can leave the model as it is.

Let's search for High Leverage points

```
hat.plot <- function(fit) {
  p <- length(coefficients(fit))
  n <- length(fitted(fit))
  plot(hatvalues(fit),
    main = "Index Plot of Hat Values"
  )
  abline(h = c(2, 3) * p / n, col = "red", lty = 2)
  identify(1:n, hatvalues(fit), names(hatvalues(fit)))
```

```
}

hat.plot(fit)
```

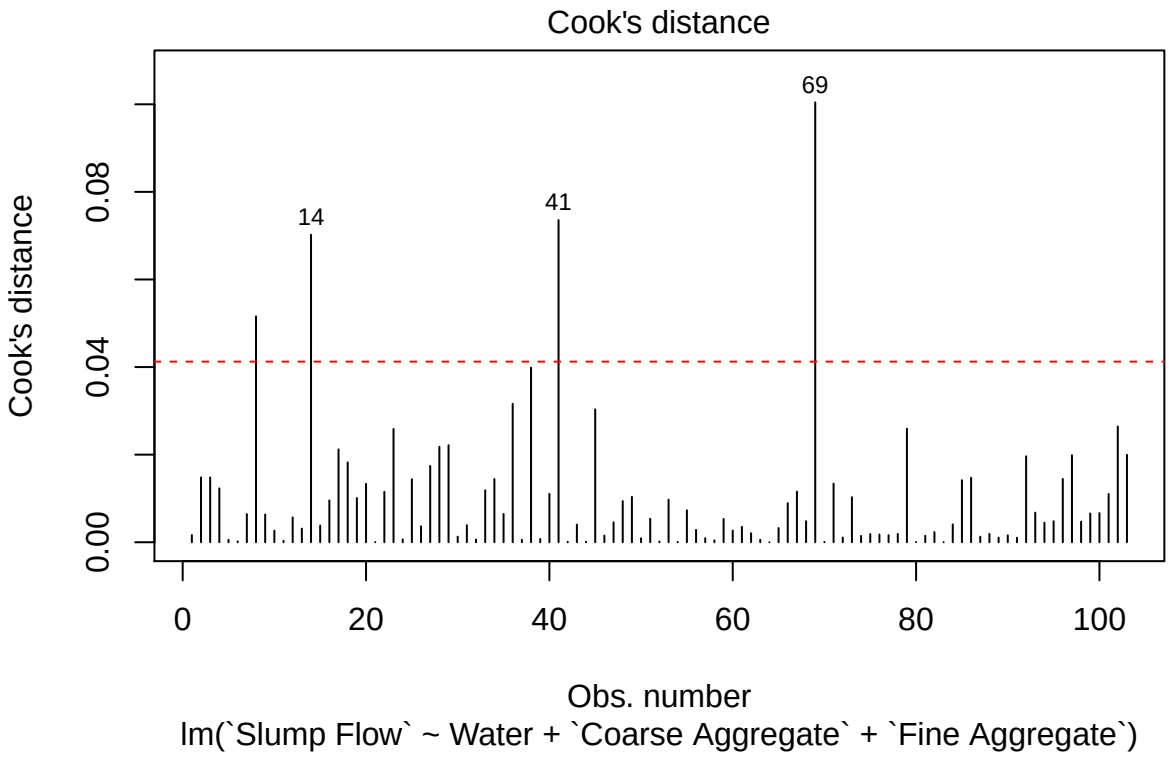



```
## integer(0)
```

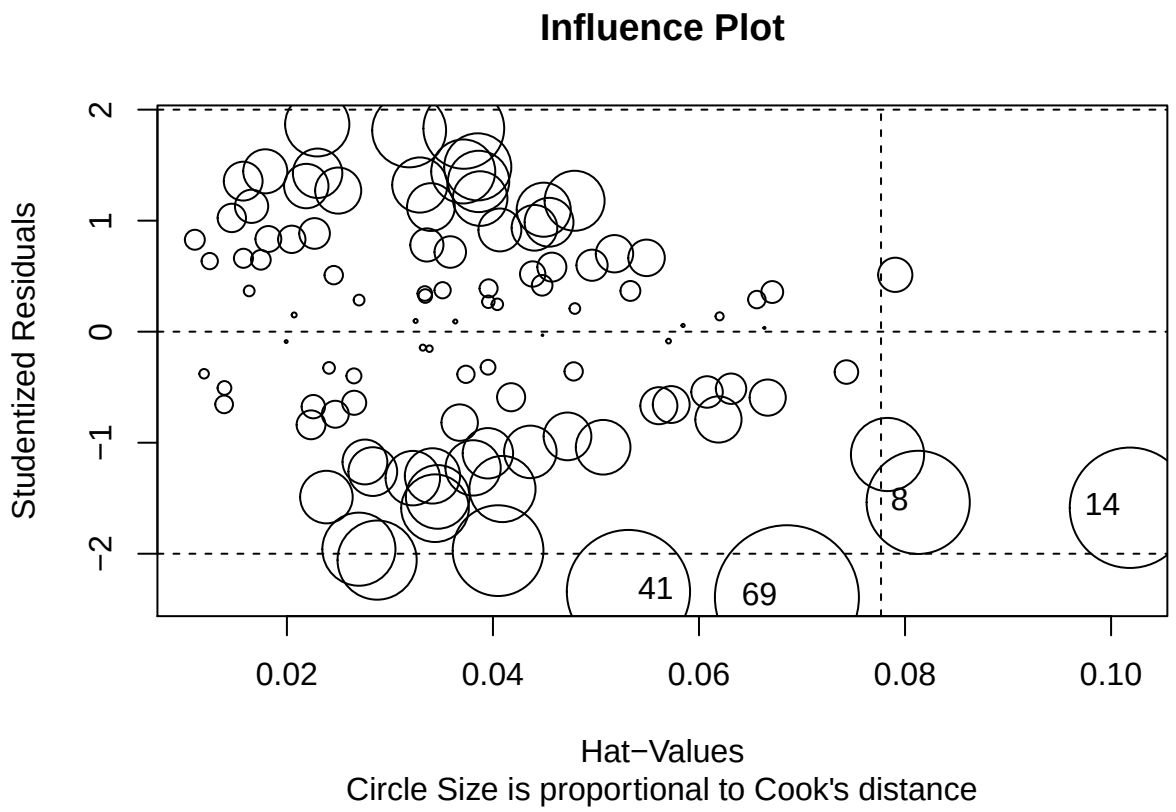
We can see that points 8, 12, 14, and 78 are unusual when it comes to their predicted values.

Influential Observations

```
cutoff <- 4 / (nrow(df) - length(fit$coefficients) - 2)
plot(fit, which = 4, cook.levels = cutoff)
abline(h = cutoff, lty = 2, col = "red")
```




```
influencePlot(fit,
  main = "Influence Plot",
  sub = "Circle Size is proportional to Cook's distance"
)
```




```
##      StudRes        Hat      CookD
## 8  -1.537566 0.08127700 0.05157597
## 14 -1.586084 0.10183117 0.07022906
## 41 -2.340784 0.05315091 0.07356555
## 69 -2.391905 0.06853364 0.10044585
```

The plot shows that 41 and 14 are outliers. 8 and 14 have high leverage. 45, 41, 8 and 14 are influential observations.

We remove points 41 and 14 as they are outliers as well as influential.

```
fit <- lm(`Slump Flow` ~ Water + `Coarse Aggregate` + `Fine Aggregate`, data = df[-c(14, 41), ])
fit2 <- lm(`Slump Flow` ~ Water + Slag + `Coarse Aggregate` + `Fine Aggregate`, data = df[-c(14, 41), ]
fit3 <- lm(`Slump Flow` ~ (Water^2) + Water + Slag, data = df[-c(14, 41), ])
```

Selecting the best regression model

```
anova(fit2, fit)
```

```
## Analysis of Variance Table
##
## Model 1: `Slump Flow` ~ Water + Slag + `Coarse Aggregate` + `Fine Aggregate`
## Model 2: `Slump Flow` ~ Water + `Coarse Aggregate` + `Fine Aggregate`
##   Res.Df   RSS Df Sum of Sq      F    Pr(>F)
## 1     96 14491
## 2     97 16353 -1   -1861.5 12.332 0.0006804 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(fit2, fit3)
```

```
## Analysis of Variance Table
##
## Model 1: `Slump Flow` ~ Water + Slag + `Coarse Aggregate` + `Fine Aggregate`
## Model 2: `Slump Flow` ~ (Water^2) + Water + Slag
##   Res.Df   RSS Df Sum of Sq      F Pr(>F)
## 1     96 14491
## 2     98 14566 -2   -74.709 0.2475 0.7813
```

```
AIC(fit, fit2, fit3)
```

```
##      df      AIC
## fit   5 810.4155
## fit2  6 800.2096
## fit3  4 796.7290
```

The p-value test tells us that fit2 is better than fit since as the Slag predictor adds extra value to our model. However it is not better than fit3 model.

The AIC test also indicated that fit3 is the best model.

Let's interpret the results

```
summary(fit3)
```

```
##
## Call:
## lm(formula = `Slump Flow` ~ (Water^2) + Water + Slag, data = df[-c(14,
##     41), ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -33.736  -9.846   1.477   9.286  23.750
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -55.78057   12.15765  -4.588 1.33e-05 ***
## Water         0.57170    0.06086   9.393 2.51e-15 ***
## Slag         -0.08749    0.02041  -4.287 4.24e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.19 on 98 degrees of freedom
## Multiple R-squared:  0.5237, Adjusted R-squared:  0.514
## F-statistic: 53.87 on 2 and 98 DF,  p-value: < 2.2e-16
```

```
predictions <- predict(fit3, df)

head(predictions)
```

```
##        1        2        3        4        5        6
## 57.10139 34.08842 33.60422 33.60422 60.19356 51.91536
```

```
rm(list = ls())
```

We can infer from the model coefficients Water is the most important predictor in calculating the value of the Slump Flow. 1 kg per M cube change in Water results to 0.57 cm change in the Slump Flow. Slag is a less important predictor.
