Homework 4 Solution

Bryan Liu

September 23, 2018

Problem 1

a) Let $X_1^{(b)}, ..., X_n^{(b)}$ be a boostrap sample of the observed data $X_1, ..., X_n$. That is, $X_1^{(b)}, ..., X_n^{(b)}$ are obtained by sampling n points from $\{X_1, ..., X_n\}$ with replacement. Compute the standard deviation using the bootstrap sample,

$$\hat{\sigma}^{(b)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i^{(b)} - \bar{X}_i^{(b)})^2} \tag{1}$$

Repeat this procedure for B times and obtain standard errors $\hat{\sigma}^{(1)}, ..., \hat{\sigma}^{(B)}$. For a 95% confidence interal, take the 2.5th and 97.5 quantiles of $\{\hat{\sigma}^{(1)}, ..., \hat{\sigma}^{(B)}\}$.

b) First write a function that implements the bootstrap confidence interval procedure described in part (a)

```
get_confidence_interval <- function(x, alpha = 0.05, n_boot = 200){
  # for a vector of observations x, get bootstrap confidence
  # interval using procedure described in part (a)

  n_obs <- length(x) # number of observations

  # matrix in which we shall store bootstrap samples
  x_boot_array <- matrix(0, nrow = n_obs, ncol = n_boot)
  for(b in 1:n_boot){
    x_boot_array[, b] <- sample(x, size= n_obs, replace = TRUE)
  }

  # compute sd for each bootstrap sample
  sd_vec <- apply(x_boot_array, 2, sd)

  # return quantiles
  return(quantile(sd_vec, probs = c(alpha / 2, 1 - alpha / 2)))
}
```

We run our experiment $M = 1000$ times and check whether we get 95% coverage:

```
M <- 1000 # number of trials
coverage_bool_array <- rep(FALSE, M)
for(m in 1:M){
  # draw new x
  x <- rnorm(100, mean = 0, sd = 1)

  # get confidence intervale
  ci <- get_confidence_interval(x)

  # check if confidence interval contains 1
  coverage_bool_array[m] <- (1.0 < ci[2]) & (1.0 > ci[1])
```

```
}

# check proportion covered
print(mean(coverage_bool_array))
```

```
## [1] 0.91
```

Problem 2

We set up the X matrix and solve for the regression coefficients.

```
# load data:
load('./twoyear.RData')
two_year_data <- data

# responses
y <- two_year_data$lwage

# construct X matrix
n_obs <- dim(two_year_data)[1]
X <- matrix(c(rep(1, n_obs),
              two_year_data$jc,
              two_year_data$univ,
            two_year_data$exper), nrow = n_obs)

# get regression coefficients
XtX <- (t(X) %*% X)
hat_beta <- solve((t(X) %*% X), t(X) %*% y)
print(hat_beta)
```

```
##              [,1]
## [1,] 1.472325579
## [2,] 0.066696719
## [3,] 0.076876249
## [4,] 0.004944224
```

```
# get estimate of standard error
hat_sigma <- sqrt(sum((y - X %*% hat_beta)**2 / (n_obs - 3)))
```

a) We test the null hypothesis $H_0 : \beta_1 - \beta_2 = 0$.

The t-statistic is

$$T = \frac{\hat{\beta}_1 - \hat{\beta}_2}{s.e(\hat{\beta}_1 - \hat{\beta}_2)} \tag{2}$$

Here,

$$s.e(\hat{\beta}_1 - \hat{\beta}_2) = \sqrt{\text{Var}(\hat{\beta}_1) + \text{Var}(\hat{\beta}_2) - 2\text{Cov}(\hat{\beta}_1, \hat{\beta}_2)} \tag{3}$$

$$= \hat{\sigma}\sqrt{[(X^T X)^{-1}]_{22} + [(X^T X)^{-1}]_{33} - 2[(X^T X)^{-1}]_{23}} \tag{4}$$

where in the second line we recalled that an estimate of the covariance matrix of $\hat{\beta}$ is given by $\hat{\sigma}^2(X^TX)^{-1}$.

We compute the t statistic:

```
XtX_inv <- solve(XtX)
se <- hat_sigma * sqrt((XtX_inv[2, 2] + XtX_inv[3, 3] - 2 * XtX_inv[2, 3]))
t <- (hat_beta[2] - hat_beta[3]) / se

cat('t-statistic: ', t)
```

```
## t-statistic:  -1.467765
```

```
cat('p-value: ', 2 * pt(-1 * abs(t), df = n_obs - 3))
```

```
## p-value:  0.1422145
```

The T-test does not reject at the 5% level.

b) We set up the two models that we shall test:

```
model1 <- lm(lwage ~ jc + univ + exper, data = two_year_data)
model2 <- lm(lwage ~ I(jc + univ) + exper, data = two_year_data)

anova(model2, model1)
```

```
## Analysis of Variance Table
##
## Model 1: lwage ~ I(jc + univ) + exper
## Model 2: lwage ~ jc + univ + exper
##   Res.Df    RSS Df Sum of Sq     F Pr(>F)
## 1   6760 1250.9
## 2   6759 1250.5  1   0.39853 2.154 0.1422
```

The F-test also does not reject at the 5% level. Note that the p-values for both the F and T tests coincide.

c) Consider the models

$$\text{lwage} = \beta_0 + \beta_1 \text{jc} + \beta_2 \text{univ} + \beta_3 \text{exper} + e \tag{5}$$

and

$$\text{lwage} = \beta_0 + \alpha_1(\text{jc} - \text{univ}) + \alpha_2(\text{jc} + \text{univ}) + \beta_3 \text{exper} + e \tag{6}$$

Then by letting $\alpha_1 = \frac{\beta_1 - \beta_2}{2}$ and $\alpha_2 = \frac{\beta_1 + \beta_2}{2}$, we see that both models are equivalent. Hence, testing $\alpha_1 = 0$ is equivalent to testing $\beta_1 - \beta_2 = 0$.

So we do a permutation test using the second formulation to test $\alpha_1 = 0$.

First, set up the X matrix for the model in equation 6

```
# Set up X Matrix
X2 <- matrix(c(rep(1, n_obs),
               two_year_data$jc - two_year_data$univ,
               two_year_data$jc + two_year_data$univ,
              two_year_data$exper), nrow = n_obs)
```

The statistic of interest is $T = \hat{\alpha}_1 / s.e.(\hat{\alpha}_1)$, and we will test whether $|T|$ is significantly different from 0 under the permutation distribution.

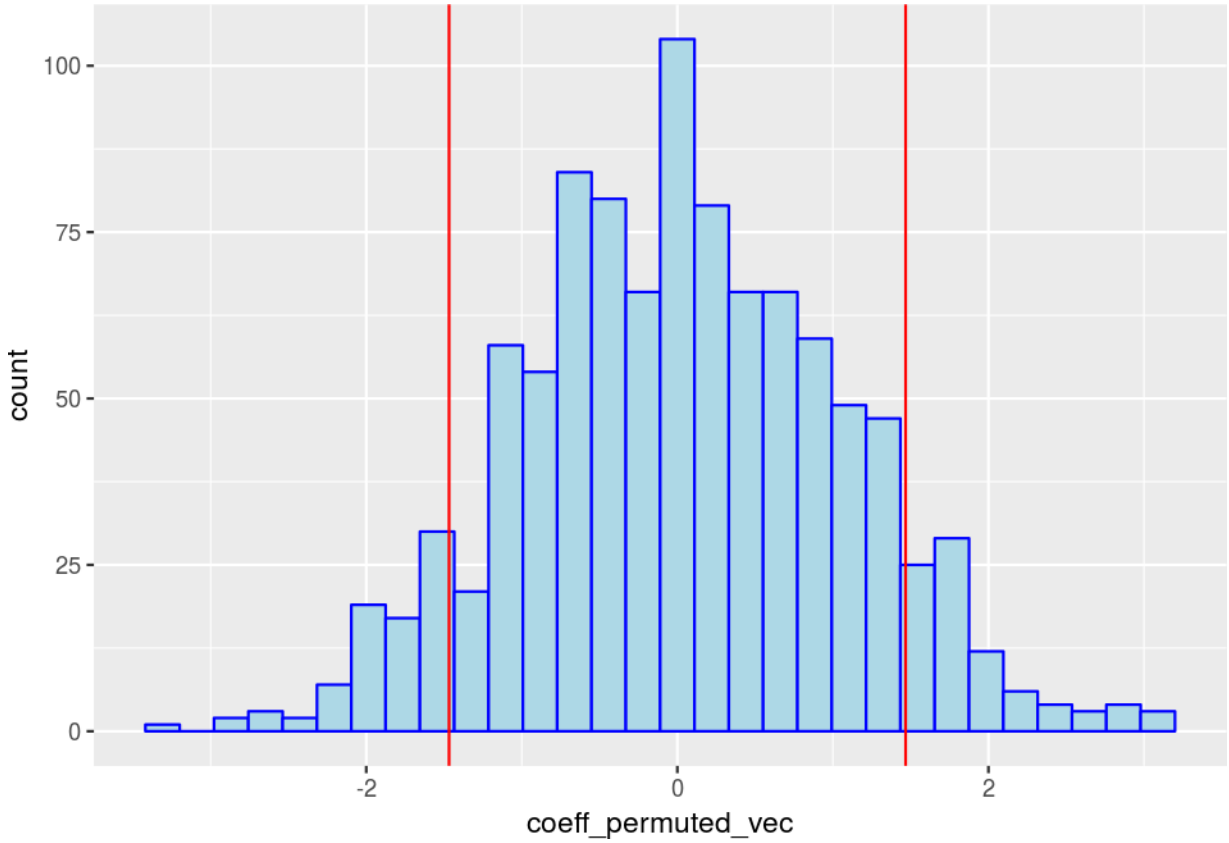
```r
get_test_stat <- function(X, y, which_coeff = 2){
  # For data X and y, returns the T statistic for
  # the \beta_{which_coeff}.

  XtX <- (t(X) %*% X)
  hat_beta <- solve((t(X) %*% X), t(X) %*% y)

  n_obs <- length(y)
  p <- dim(X)[2]

  hat_sigma <- sqrt(sum((y - X %*% hat_beta)**2 / (n_obs - p)))

  XtX_inv <- solve(XtX)

  se <- hat_sigma * sqrt(XtX_inv[which_coeff, which_coeff])

  return(hat_beta[which_coeff] / se)
}

og_test_stat <- get_test_stat(X2, y, 2)

# this is the original test statistic
print(og_test_stat)
```

```
## [1] -1.467657
```

We now run the permuatation test

```r
# function to permute columns and return coefficient
permute_column_get_coeff <- function(X, y, which_column){
  # which_column specifies the column of X to permute
  n_obs <- length(y)

  # sample observations
  sample_indx <- sample(n_obs, size = n_obs, replace = FALSE)

  # NOTE: we only permute the one column of X
  X_sampled <- X
  X_sampled[, which_column] <- X[sample_indx, which_column]

  # get regression coefficients
  # coeff_permuted <- get_regression_coefficients(X_sampled , y)[which_column]
  coeff_permuted <- get_test_stat(X_sampled, y, which_column)
  return(coeff_permuted)
}

# run the permutation test
n_perms <- 1000
coeff_permuted_vec <- rep(0, n_perms)
for(i in 1:n_perms){
  # note that we only permute the second column
  coeff_permuted_vec[i] <- permute_column_get_coeff(X2, y, which_column = 2)
}

ggplot() + geom_histogram(aes(x = coeff_permuted_vec),
```

```
                          color = 'blue', fill = 'light blue') +
        geom_vline(xintercept = og_test_stat, color = 'red') +
        geom_vline(xintercept = -og_test_stat, color = 'red')
```




```
# check p-value
mean(coeff_permuted_vec > -og_test_stat) + mean(coeff_permuted_vec < og_test_stat)
```

```
## [1] 0.158
```

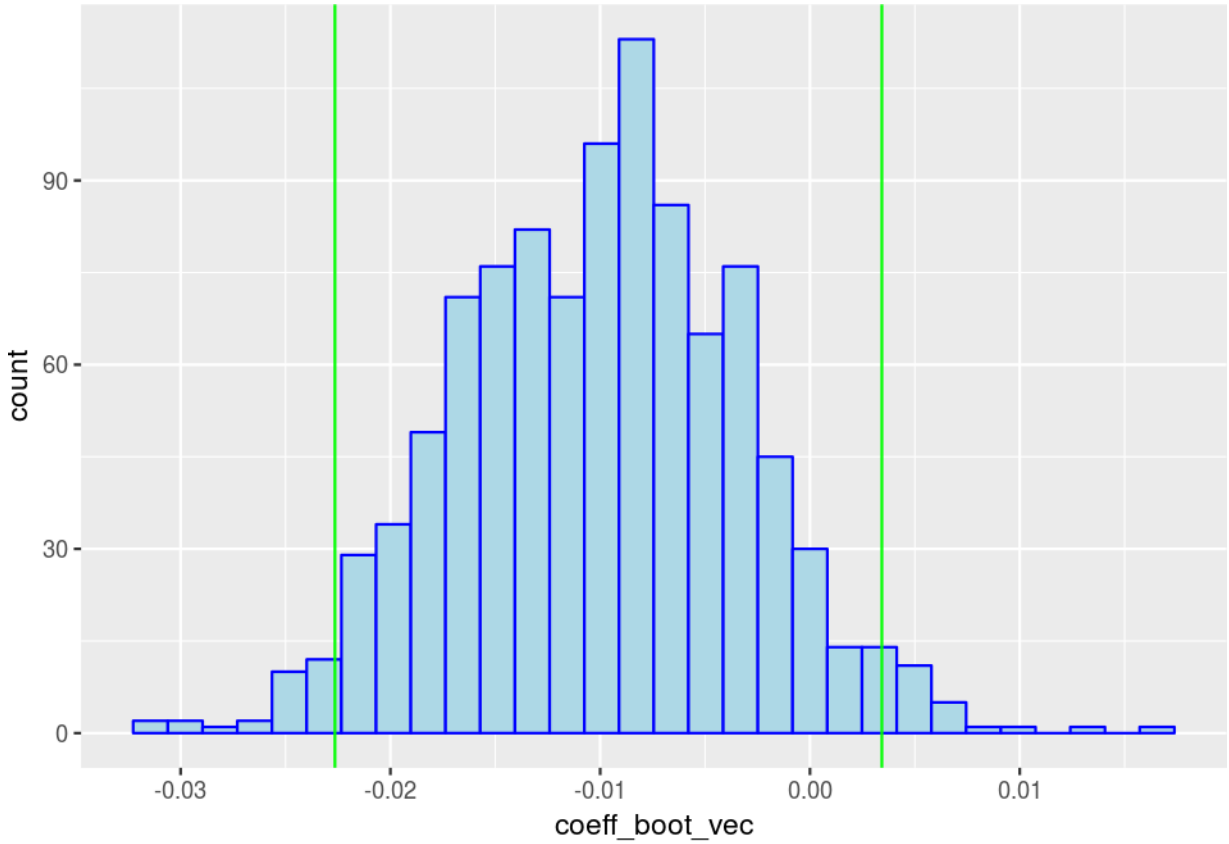
d)

```
# function to sample data (x, y) and return coefficient
get_bootstrap_coeff <- function(X, y){
  n_obs <- length(y)

  # sample observations
  sample_indx <- sample(n_obs, size = n_obs, replace = TRUE)

  X_sampled <- X[sample_indx, ]
  y_sampled <- y[sample_indx]

  # get regression coefficients
  coeff_bootstrap <- solve(t(X_sampled) %*% X_sampled,
                           t(X_sampled) %*% y_sampled)
  return(coeff_bootstrap)
}

# run the bootstrap
```

```
n_boot <- 1000
coeff_boot_vec <- rep(0, n_perms)
for(i in 1:n_boot){
  coeff_boot_vec[i] <- 2 * get_bootstrap_coeff(X2, y)[2]
}
```

We get a bootstrap confidence interval by taking the 2.5th and 97.5th quantile of the bootstrap distribution. These quantiles are shown in green below:

```
ggplot() + geom_histogram(aes(x = coeff_boot_vec),
                          color = 'blue', fill = 'light blue') +
  geom_vline(xintercept = quantile(coeff_boot_vec, 0.975), color = 'green') +
  geom_vline(xintercept = quantile(coeff_boot_vec, 0.025), color = 'green')
```



It contains 0.

NOTE: we ran the bootstrap for the model in equation 6, to get a confidence interval for $\hat{\alpha}_1$. Multiplying by 2 gave us a confidence interval for $\hat{\beta}_1 - \hat{\beta}_2$.

Problem 3

We load the savings dataset and fit a model

```
library(faraway)
data(savings)
# savings
```

```r
model <- lm(sr ~ pop15 + pop75 + dpi + ddpi, data = savings)
n_obs <- dim(savings)[1]
```

a). Lets take a look at the model

```r
summary(model)
```

```
##
## Call:
## lm(formula = sr ~ pop15 + pop75 + dpi + ddpi, data = savings)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.2422 -2.6857 -0.2488  2.4280  9.7509
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 28.5660865  7.3545161   3.884 0.000334 ***
## pop15       -0.4611931  0.1446422  -3.189 0.002603 **
## pop75       -1.6914977  1.0835989  -1.561 0.125530
## dpi         -0.0003369  0.0009311  -0.362 0.719173
## ddpi         0.4096949  0.1961971   2.088 0.042471 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.803 on 45 degrees of freedom
## Multiple R-squared:  0.3385, Adjusted R-squared:  0.2797
## F-statistic: 5.756 on 4 and 45 DF,  p-value: 0.0007904
```

A normality based confidence interval for each coefficient is given by

$$\hat{\beta}_i \pm t_{n-2}^{0.975} \text{s.e.}(\beta_i) \tag{7}$$

```r
model_summary <- summary(model)

# get estimates:
coeff <- model_summary$coefficients[, 'Estimate']

# get se's
se <- model_summary$coefficients[, 'Std. Error']

# get t stat
t <- qt(0.975, df = dim(savings)[1] - 2)

# upper confidence bound:
print(coeff + se * t)
```

```
##  (Intercept)        pop15        pop75          dpi         ddpi
## 43.353332249 -0.170370463  0.487223997  0.001535215  0.804175692
```

```r
# lower confidence bound:
print(coeff - se * t)
```

```
##  (Intercept)        pop15        pop75          dpi         ddpi
## 13.778840832 -0.752015832 -3.870219350 -0.002209018  0.015214164
```

```r
# set-up X matrix
X <- matrix(c(rep(1, n_obs),
              savings$pop15,
              savings$pop75,
              savings$dpi,
              savings$ddpi), nrow = n_obs)
y <- savings$sr

# run the bootstrap
n_boot <- 1000
coeff_boot_array <- matrix(0, nrow = 5, ncol = n_boot)
for(i in 1:n_boot){
  # note that we only permute the second column
  coeff_boot_array[, i] <- get_bootstrap_coeff(X, y)
}

# upper boostrap confidence bound
apply(coeff_boot_array, 1, quantile, probs = 0.975)
```

```
## [1] 40.446488142 -0.126551535  0.619440633  0.001245997  1.053265573
```

```r
# lower bootstrap confidence bound
apply(coeff_boot_array, 1, quantile, probs = 0.025)
```

```
## [1] 11.121081647 -0.697232748 -3.835421171 -0.001345611  0.111453296
```

Problem 4

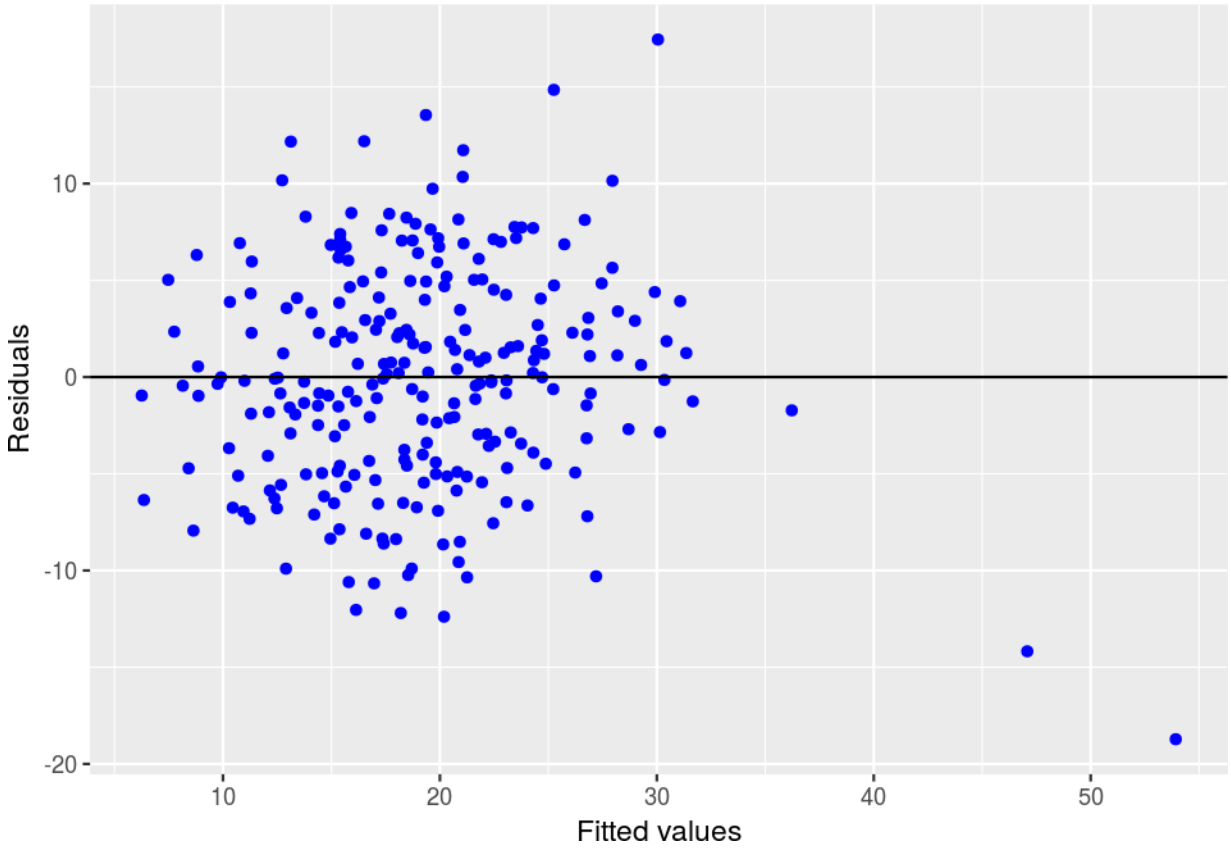
```r
# load data:
bodyfat <- read.csv('./Bodyfat.csv')
names(bodyfat)
```

```
##  [1] "Density" "bodyfat" "Age"     "Weight"  "Height"  "Neck"    "Chest"
##  [8] "Abdomen" "Hip"     "Thigh"   "Knee"    "Ankle"   "Biceps"  "Forearm"
## [15] "Wrist"
```

```r
bodyfat_model <- lm(bodyfat ~ Age + Weight + Height + Thigh, data = bodyfat)

n_obs <- dim(bodyfat)[1]
```

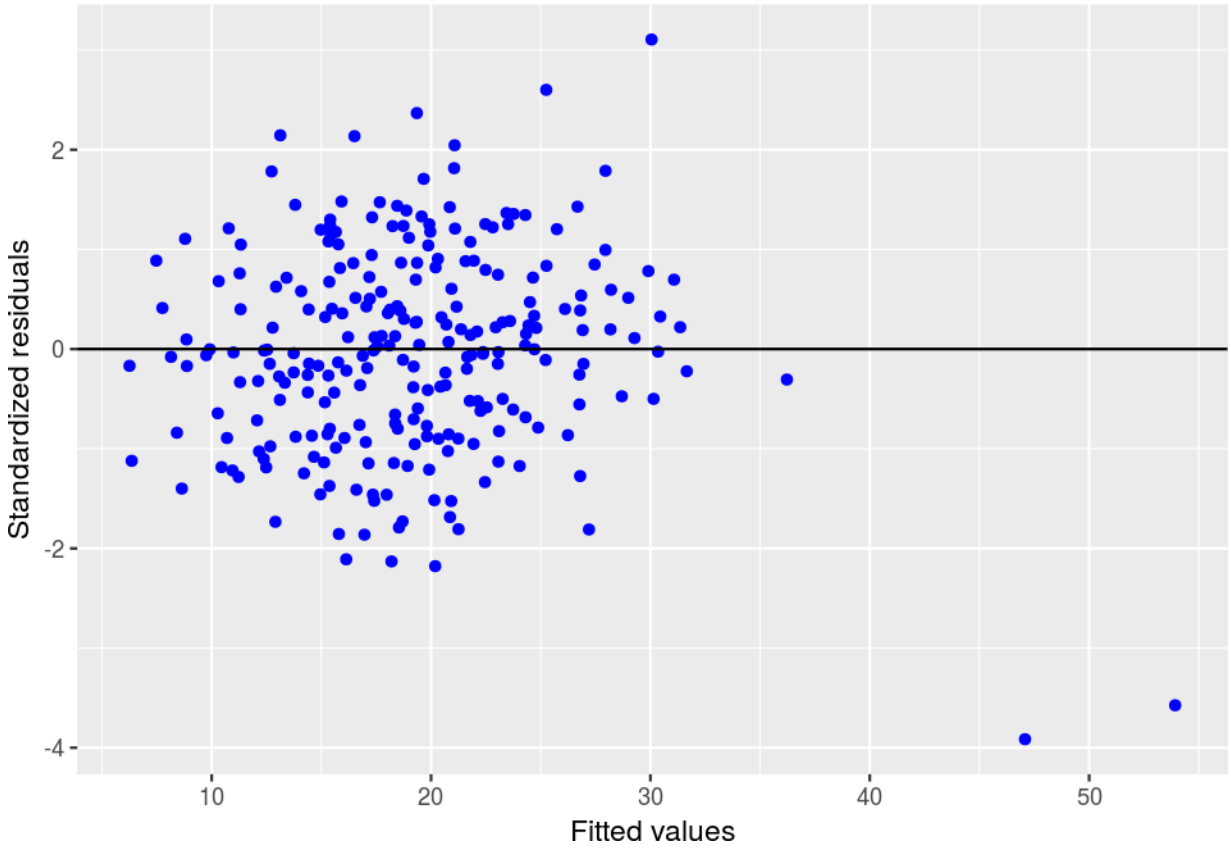
a) Residuals against fitted values

```r
# get fitted values
yhat <- predict(bodyfat_model,newdata = bodyfat)
residuals <- bodyfat$bodyfat - yhat

ggplot() + geom_point(aes(x = yhat,y = residuals), color = 'blue') +
  geom_hline(yintercept = 0) +
  xlab('Fitted values') + ylab('Residuals')
```



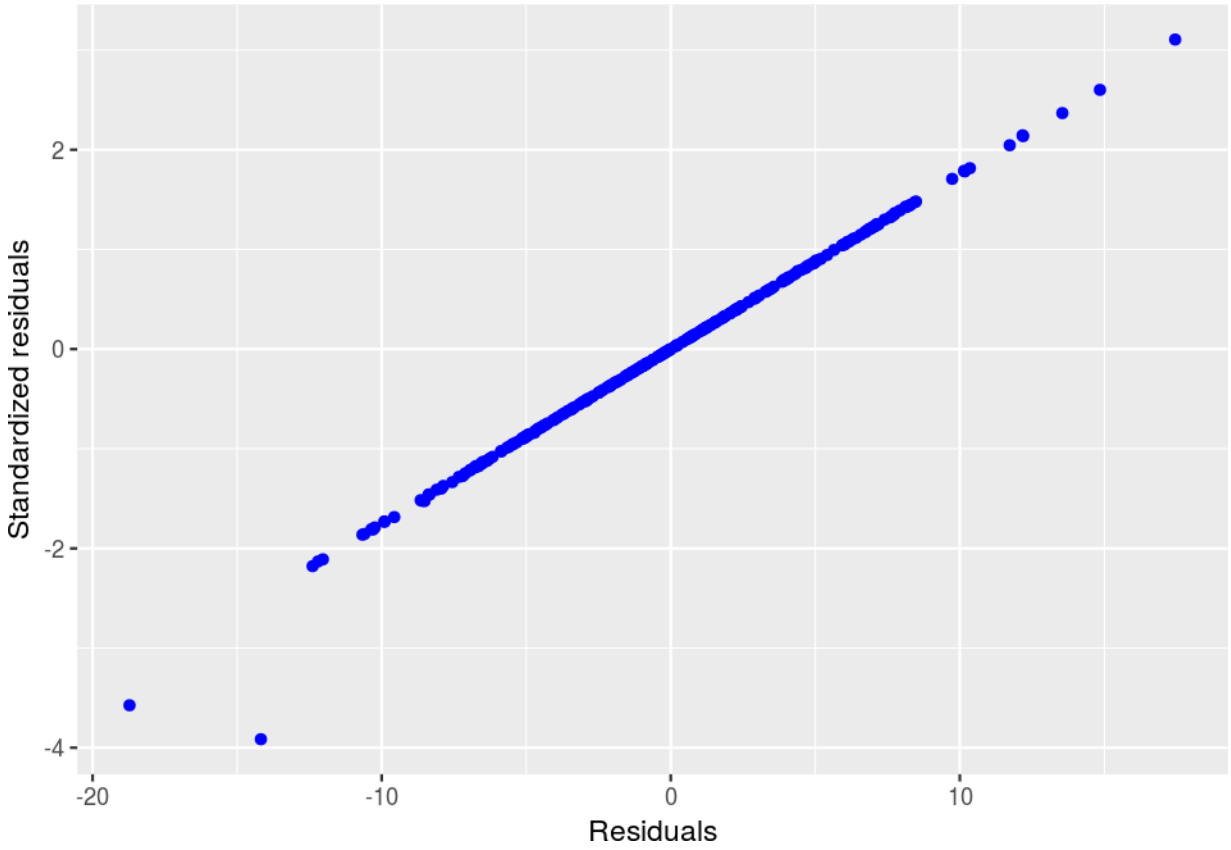
b) Standardized Residuals against fitted values

```r
# get hat matrix
X <- model.matrix(bodyfat_model)
H <- X %*% solve((t(X) %*% X), t(X))
leverages <- diag(H)

# get \hat\sigma
hat_sigma <- sqrt(sum(residuals**2) / (n_obs - 5))

# get standardized residuals
std_resid <- residuals / (hat_sigma * sqrt(1 - leverages))

ggplot() + geom_point(aes(x = yhat, y = std_resid), color = 'blue') +
  geom_hline(yintercept = 0) +
  xlab('Fitted values') + ylab('Standardized residuals')
```



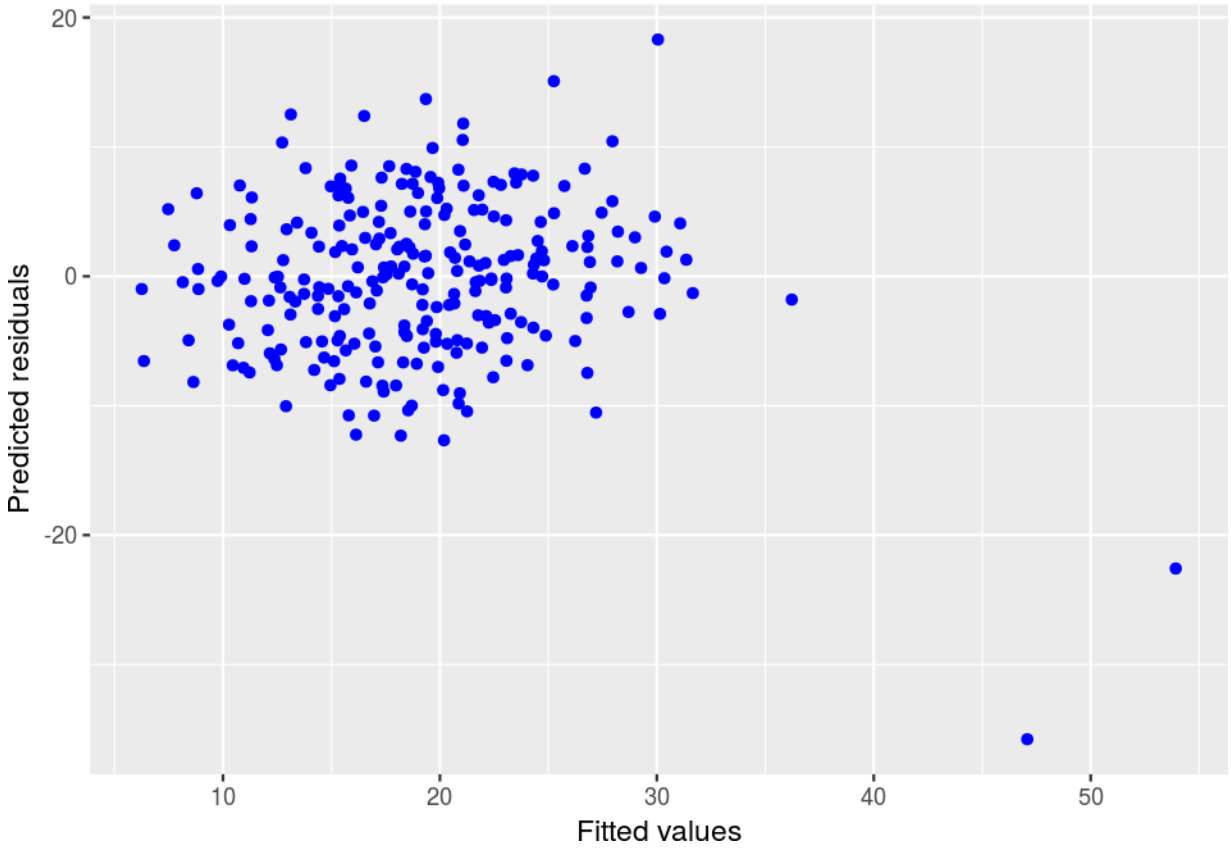
c) Residuals against Standardized Residuals.

```
ggplot() + geom_point(aes(x = residuals, y = std_resid), color = 'blue') +
  xlab('Residuals') + ylab('Standardized residuals')
```



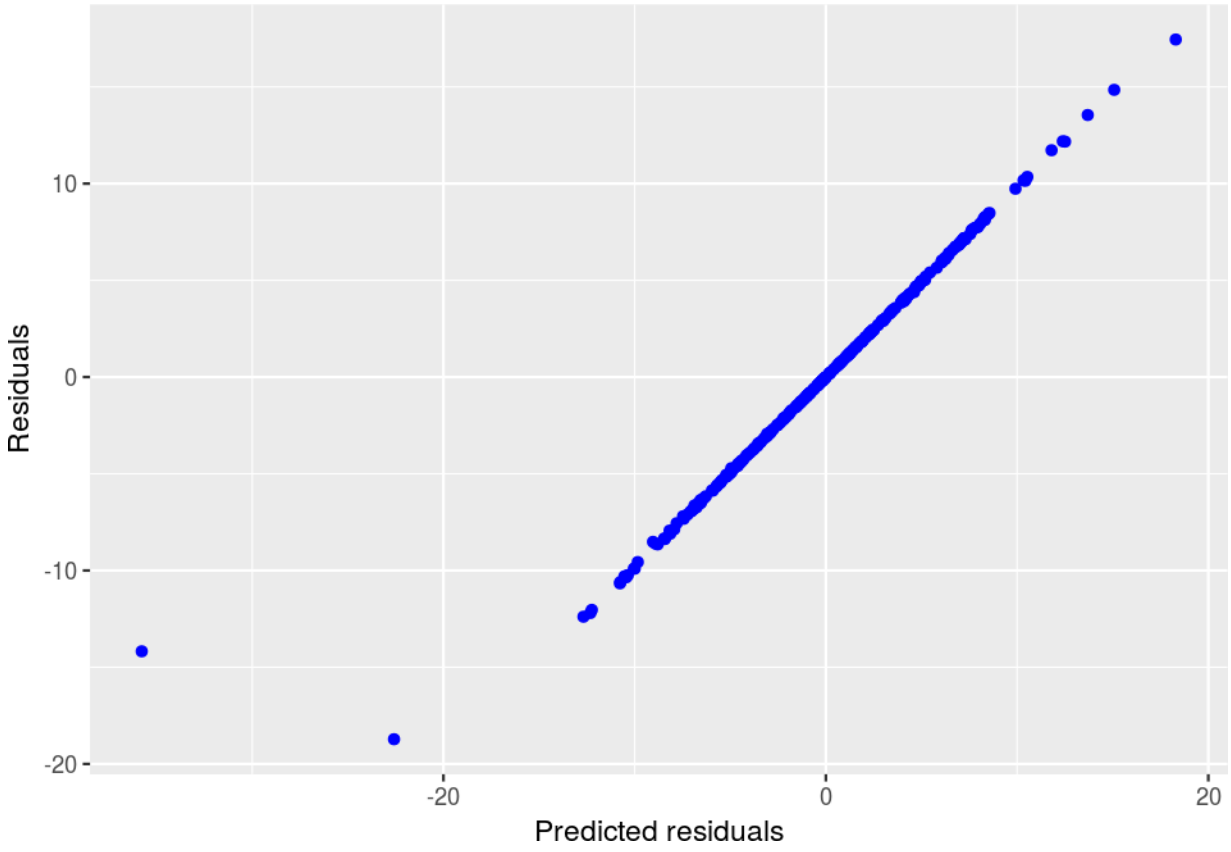
d) Predicted residuals against fitted values

```
# get predicted residuals
pred_resid <- residuals / (1 - leverages)

ggplot() + geom_point(aes(x = yhat, y = pred_resid), color = 'blue') +
  xlab('Fitted values') + ylab('Predicted residuals')
```



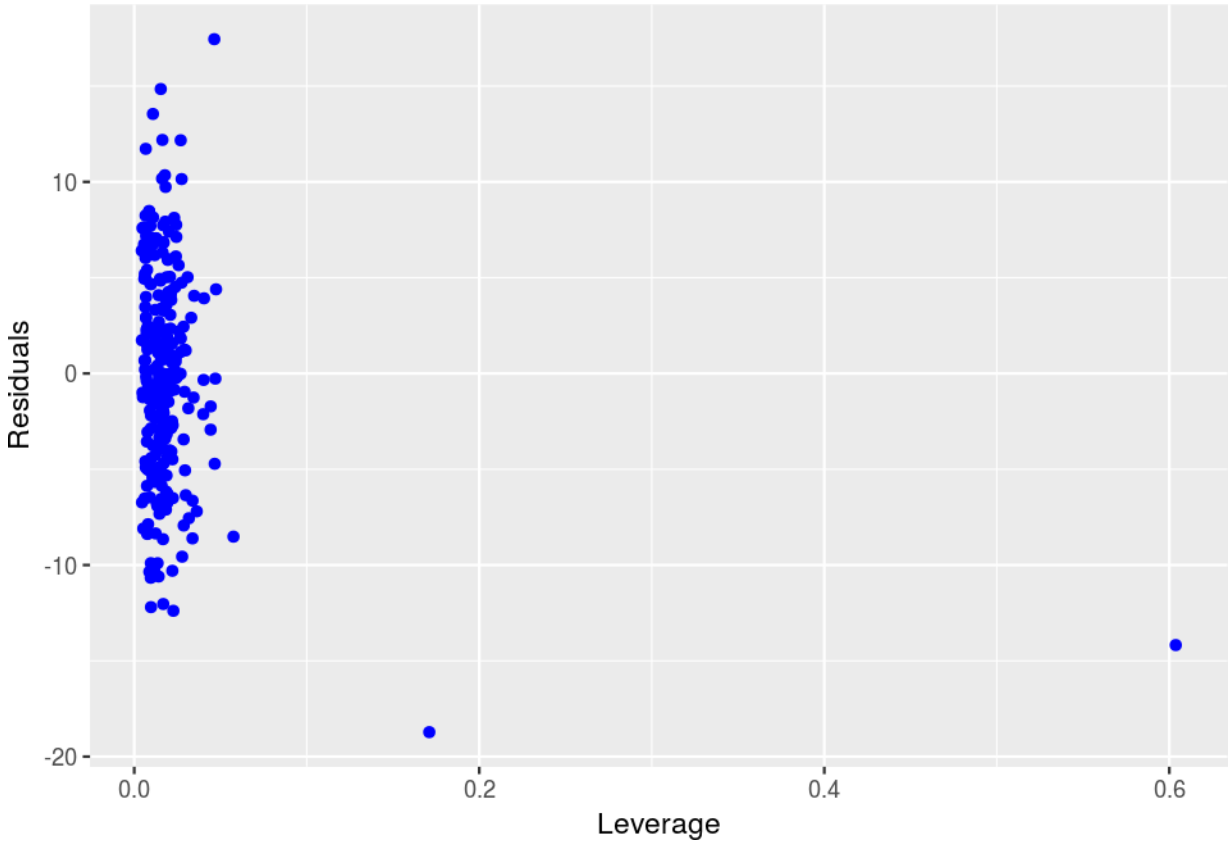
e) Residuals against predicted residuals.

```
ggplot() + geom_point(aes(x = pred_resid, y = residuals), color = 'blue') +
  xlab('Predicted residuals') + ylab('Residuals')
```



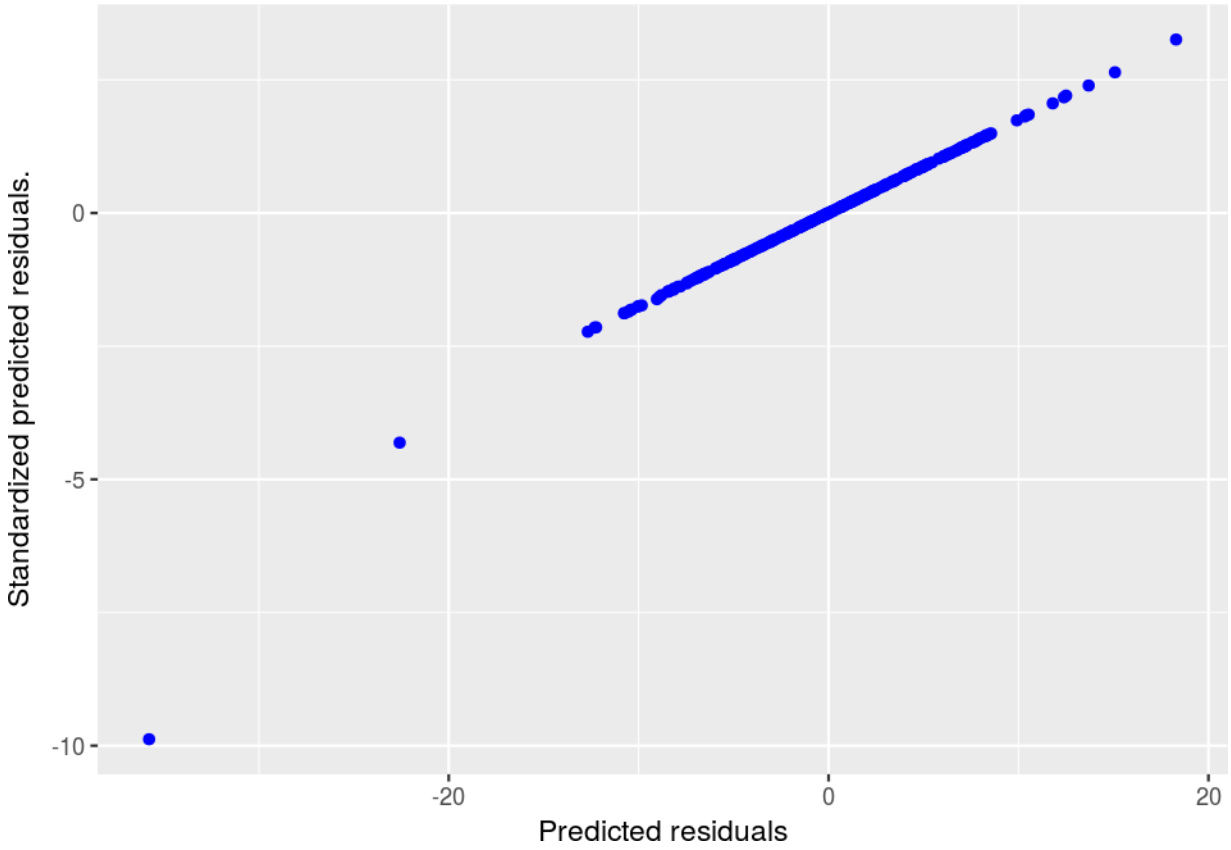
f) Residuals against leverage.

```
ggplot() + geom_point(aes(x = leverages, y = residuals), color = 'blue') +
  xlab('Leverage') + ylab('Residuals')
```



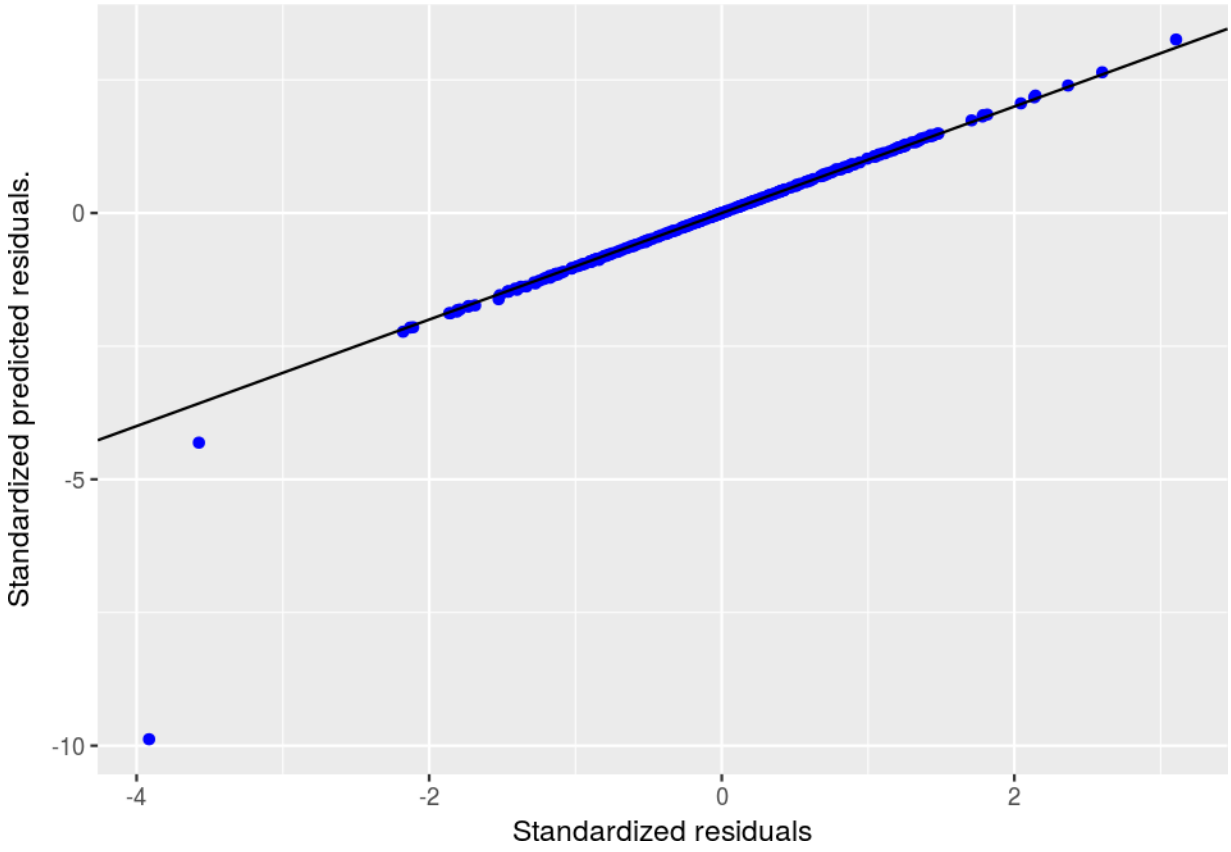
g) Predicted residuals against Standardized Predicted Residuals.

```
# get standardized predicted residuals
std_pred_resid <- pred_resid / (hat_sigma * sqrt(1 - leverages))

ggplot() + geom_point(aes(x = pred_resid, y = std_pred_resid), color = 'blue') +
  xlab('Predicted residuals') + ylab('Standardized predicted residuals.')
```



h) Standardized residuals against Standardized Predicted residuals.

```
ggplot() + geom_point(aes(x = std_resid, y = std_pred_resid), color = 'blue') +
  geom_abline(slope = 1) +
  xlab('Standardized residuals') + ylab('Standardized predicted residuals.')
```




```
cooks<- cooks.distance(model = bodyfat_model)
plot(cooks)
```