Assignment 4

Daniel Lee

3/17/2017

I collaborated with Josiah Davis. I received a lot of coding help from him.

Problem 2

```
rm(list = ls())
library(MPV)
```

```
## Warning: package 'MPV' was built under R version 3.3.2
```

```
##
## Attaching package: 'MPV'
```

```
## The following object is masked from 'package:datasets':
##
##     stackloss
```

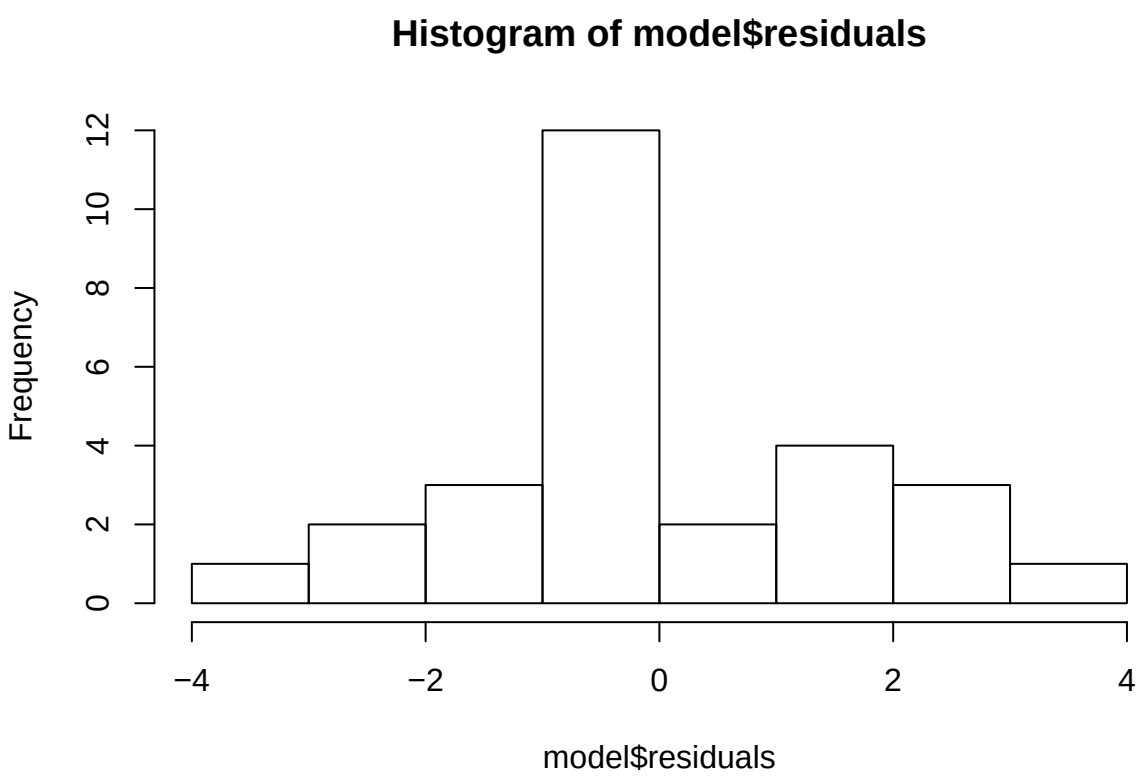
```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

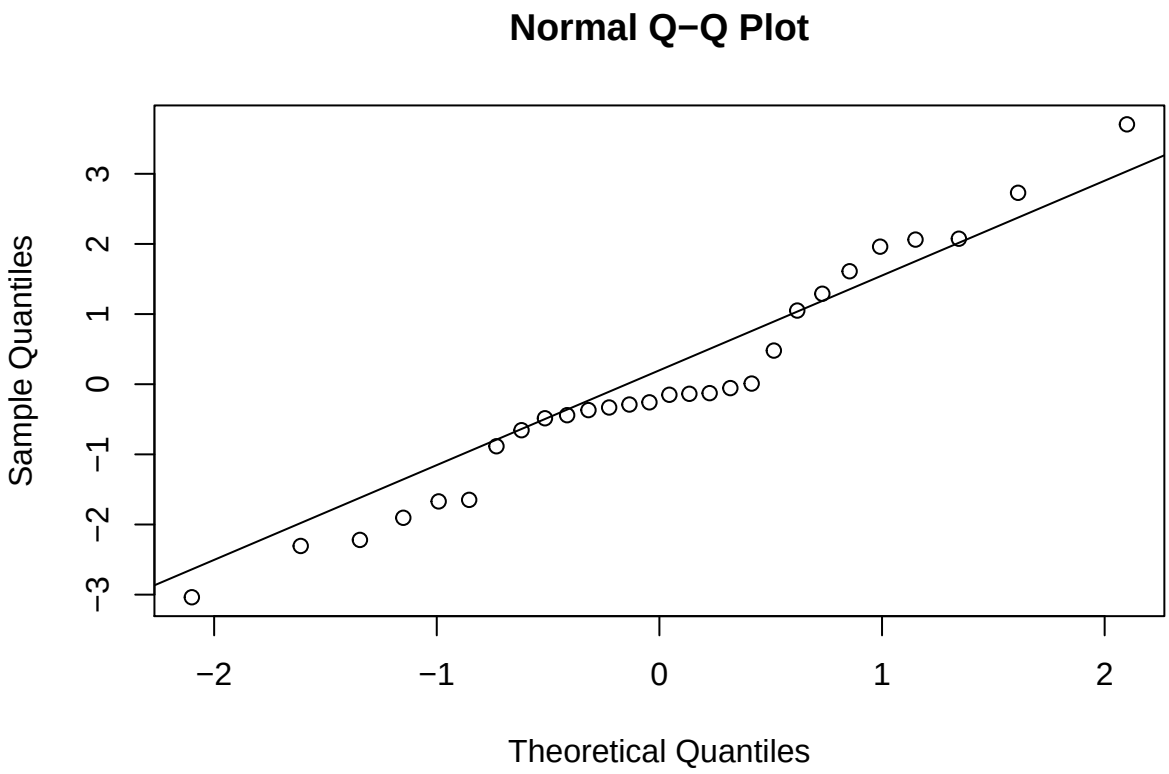
```
library(reshape2)

data(table.b1)
nfl <- table.b1
```

a: Construct a normal probability plot of the residuals. Does there seem to be any problem with the normality assumption?

```
model <- lm(y ~ x2 + x7 + x8, data = nfl)
hist(model$residuals, breaks = 8)
```



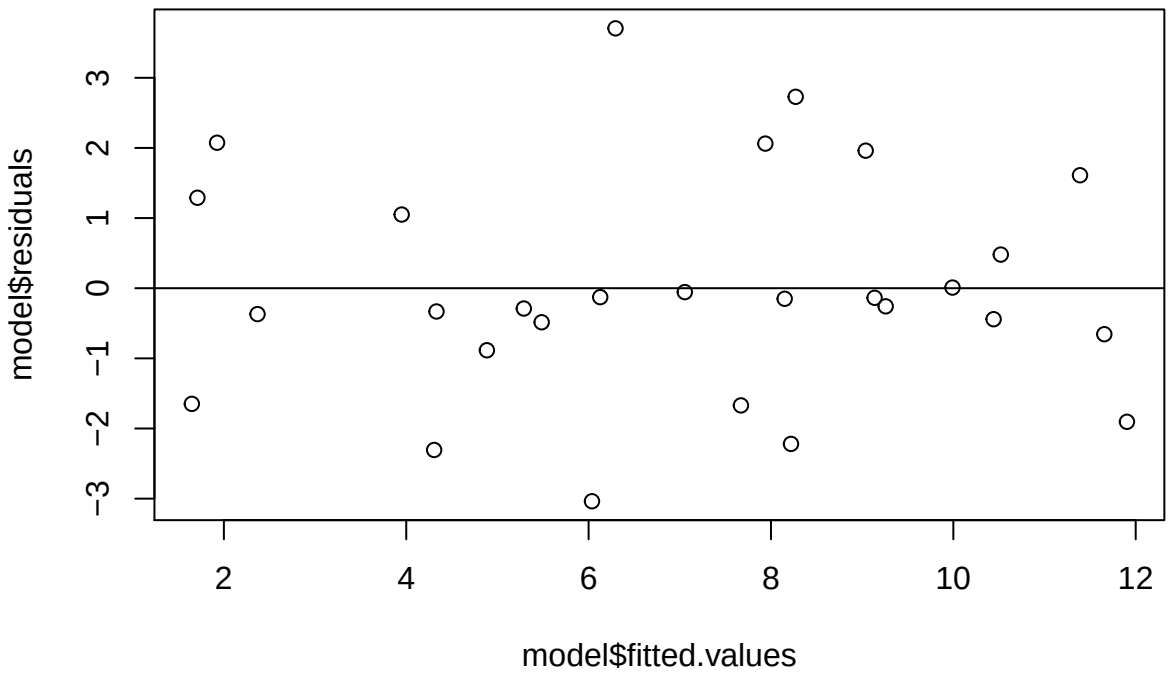

```
qqnorm(model$residuals)
qqline(model$residuals)
```



From the q-q plot, there seems to be a problem with the normality assumption. This can be observed by the residuals that are either greatly below or greatly above the line. Particularly in the middle of the plot, the sample quantiles seem to deviate from the theoretical quantiles. From the histogram, the distribution seems to be too concentrated in the middle to be a normal distribution.

b. Construct and interpret a plot of the residuals versus the predicted response.

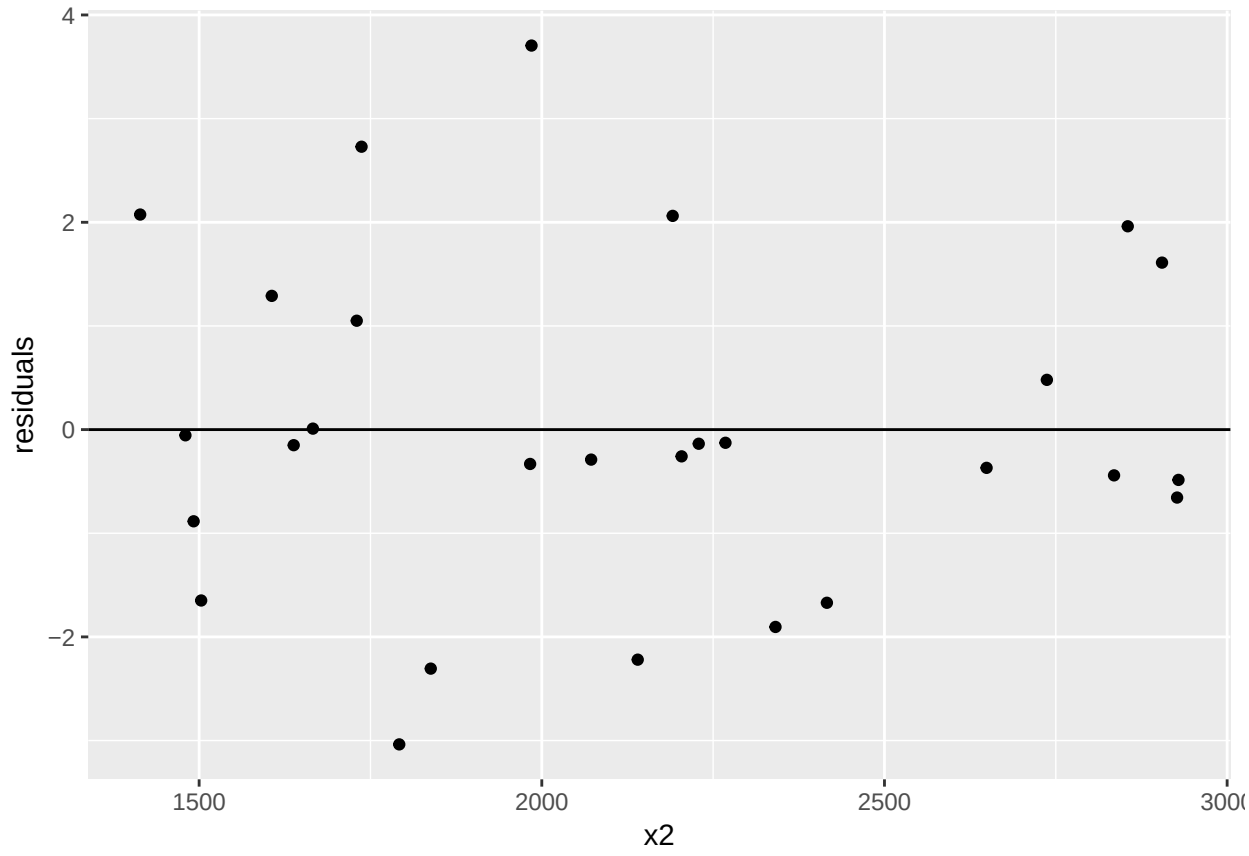
```
plot(model$fitted.values, model$residuals)
abline(h = 0)
```



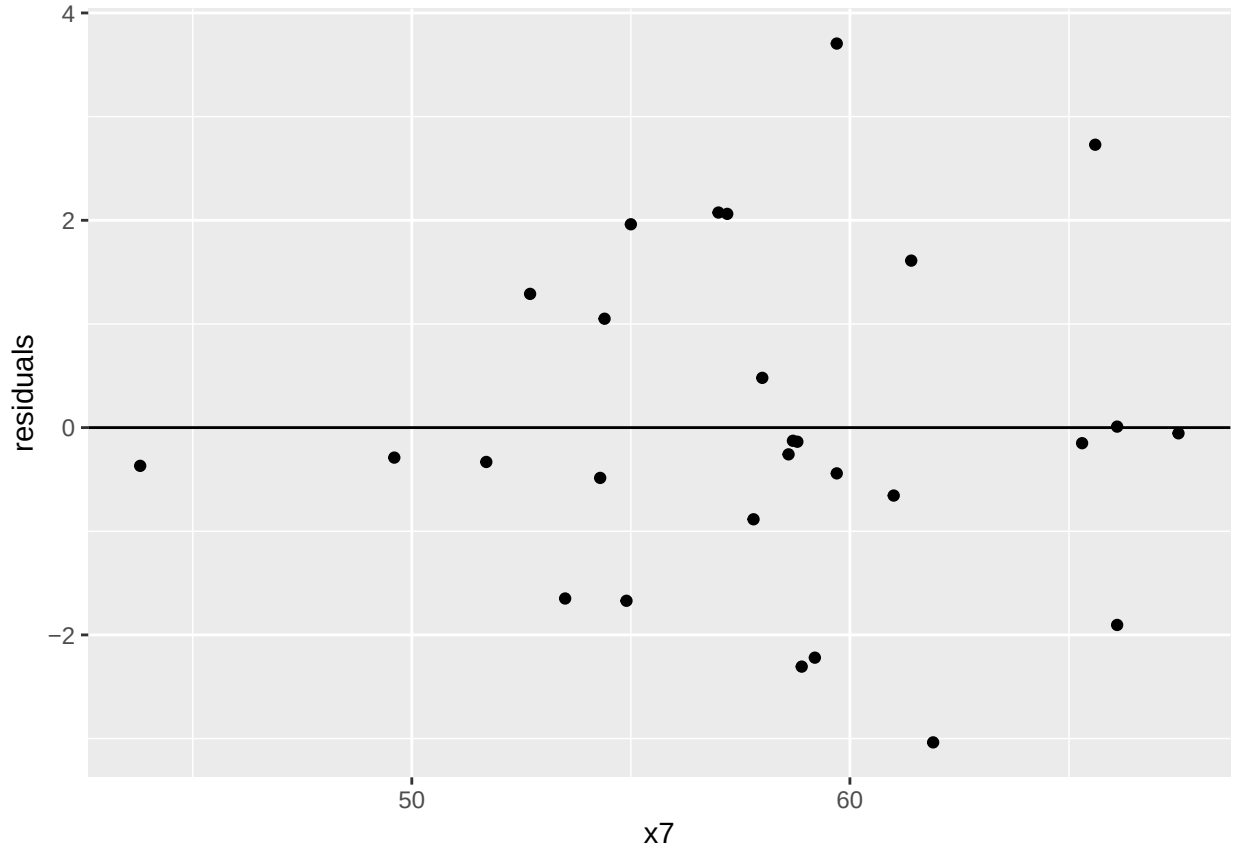
It is not clear from this plot whether a pattern of heterskedasticity is present, although this does not seem to be the case. Some of the values may be outliers, but it is unclear without studentization. The scatter seems to be random, so there is no apparent need for additional data transormations of the input variables.

c. Construct plots of the residuals versus each of the regressor variables. Do these plots imply that the regressor is correctly specified?

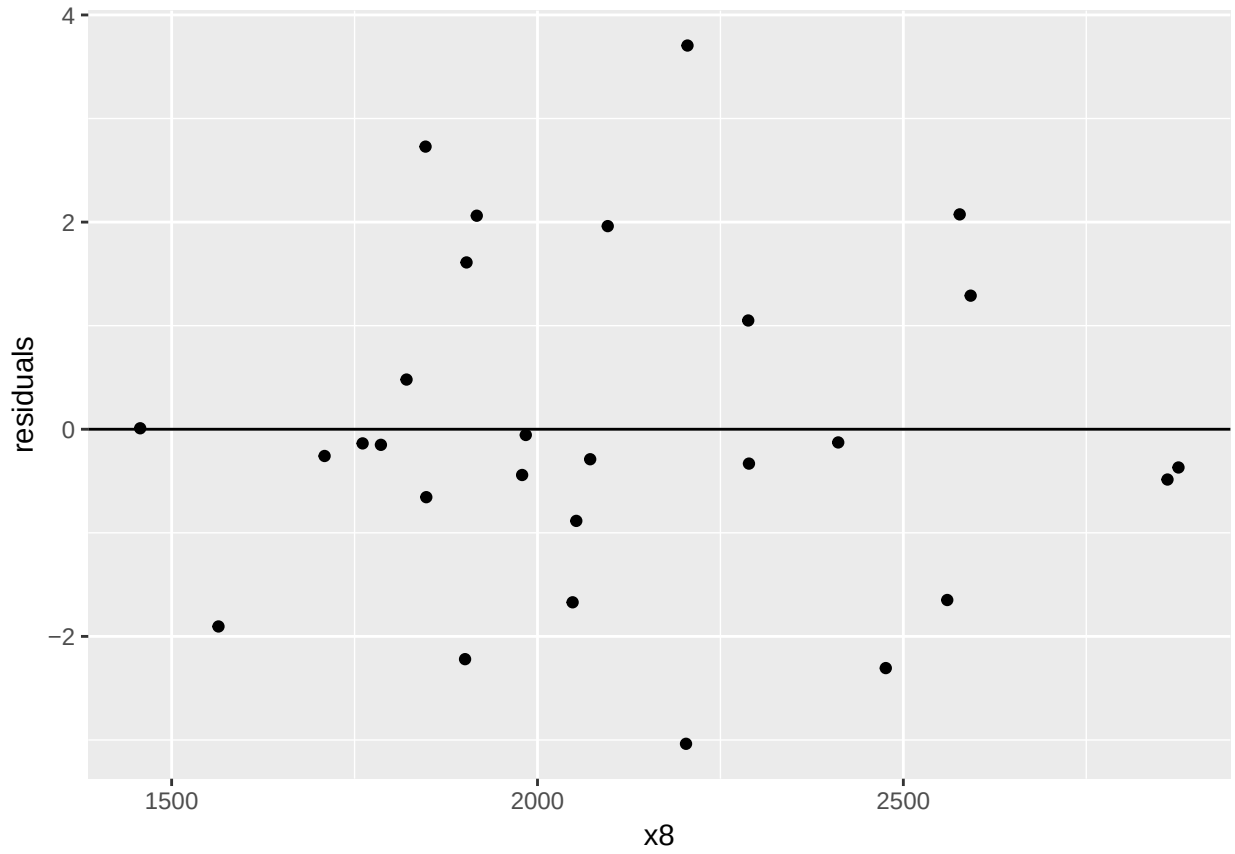
```
nfl$fitted <- model$fitted.values
nfl$residuals <- model$residuals

plot_c <- ggplot(nfl, aes(y = residuals)) + geom_hline(yintercept = 0)
plot_c + geom_point(aes(x = x2))
```




```
plot_c + geom_point(aes(x = x7))
```




```
plot_c + geom_point(aes(x = x8))
```



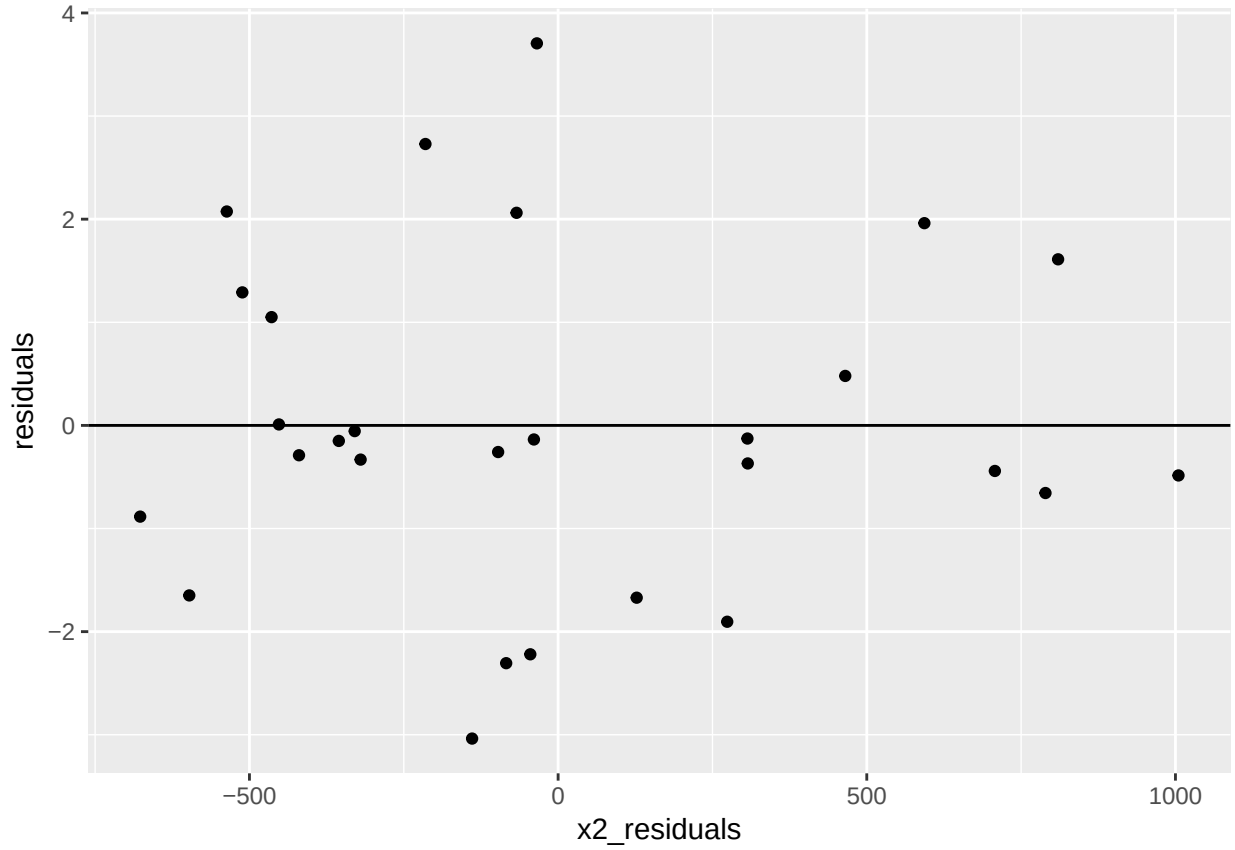
The residual vs. regressor plots do not imply that the regression model is correctly specified. This is due to the apparent presence of heteroskedasticity in the plotting of x2 and x7. There is no obvious heteroskedasticity in the plot of x8. There are no other obvious patterns aside from the heteroskedasticity.

d. Construct the partial regression plots for this model. Compare the plots with the plots of residuals versus regressors from part c above. Discuss the type of information provided by these plots.

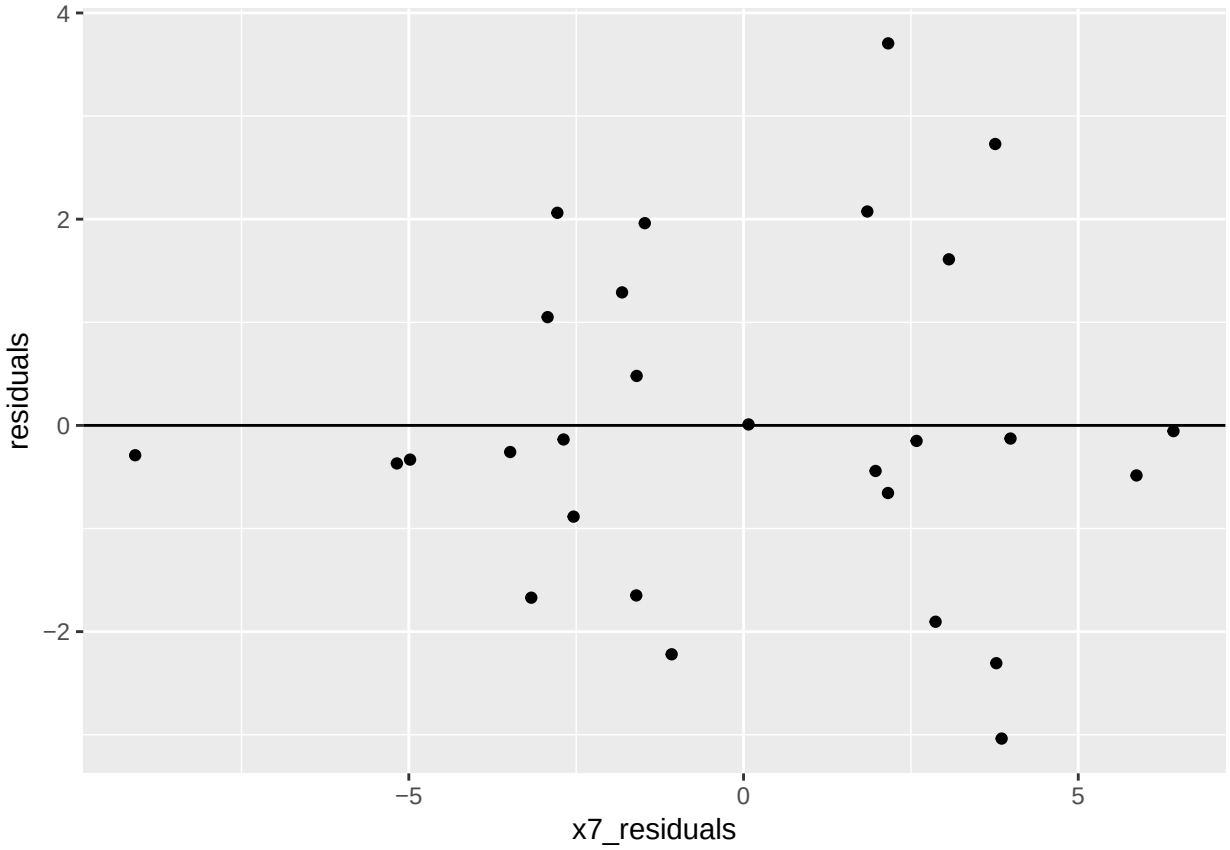
A partial regression plot is the amount of variance unexplained by the full model on the y-axis vs. the x_2 that is not contained in the particular regresser. Here are the steps:

1. Calculate the residuals for the full model. This will be the y-axis.
2. Calculate the residuals for the model regressing x_2 on x_7 and x_8. This will be the x-axis.

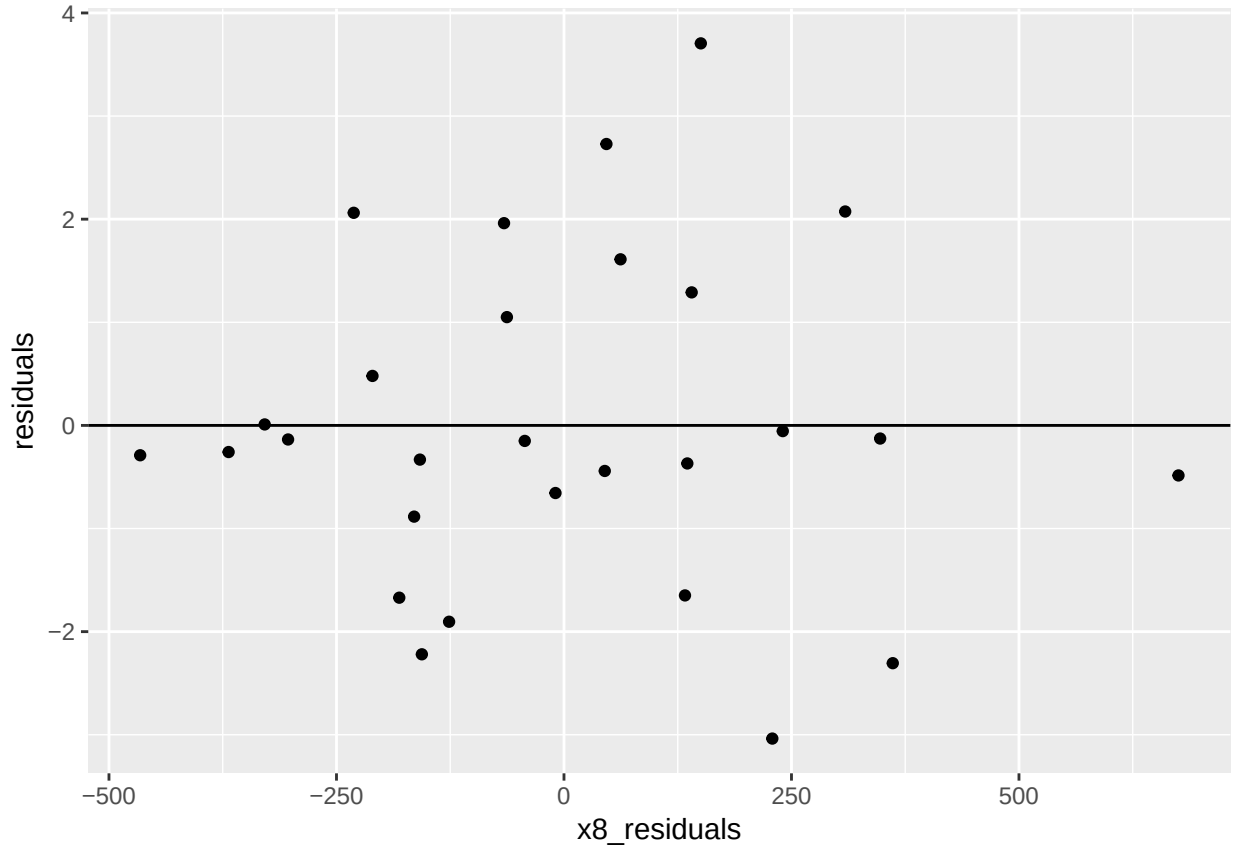
```
# Get the amount of x2 not explained by x7 and x8
nfl$x2_residuals <- lm(x2 ~ x7 + x8, data = nfl)$residuals

# Get the amount of x7 not explained by x2 and x8
nfl$x7_residuals <- lm(x7 ~ x2 + x8, data = nfl)$residuals

# Get the amount of x8 not explained by x2 and x7
nfl$x8_residuals <- lm(x8 ~ x2 + x7, data = nfl)$residuals

plot_d <- ggplot(nfl, aes(y = residuals)) + geom_hline(yintercept = 0)
plot_d + geom_point(aes(x = x2_residuals))
```




```
plot_d + geom_point(aes(x = x7_residuals))
```



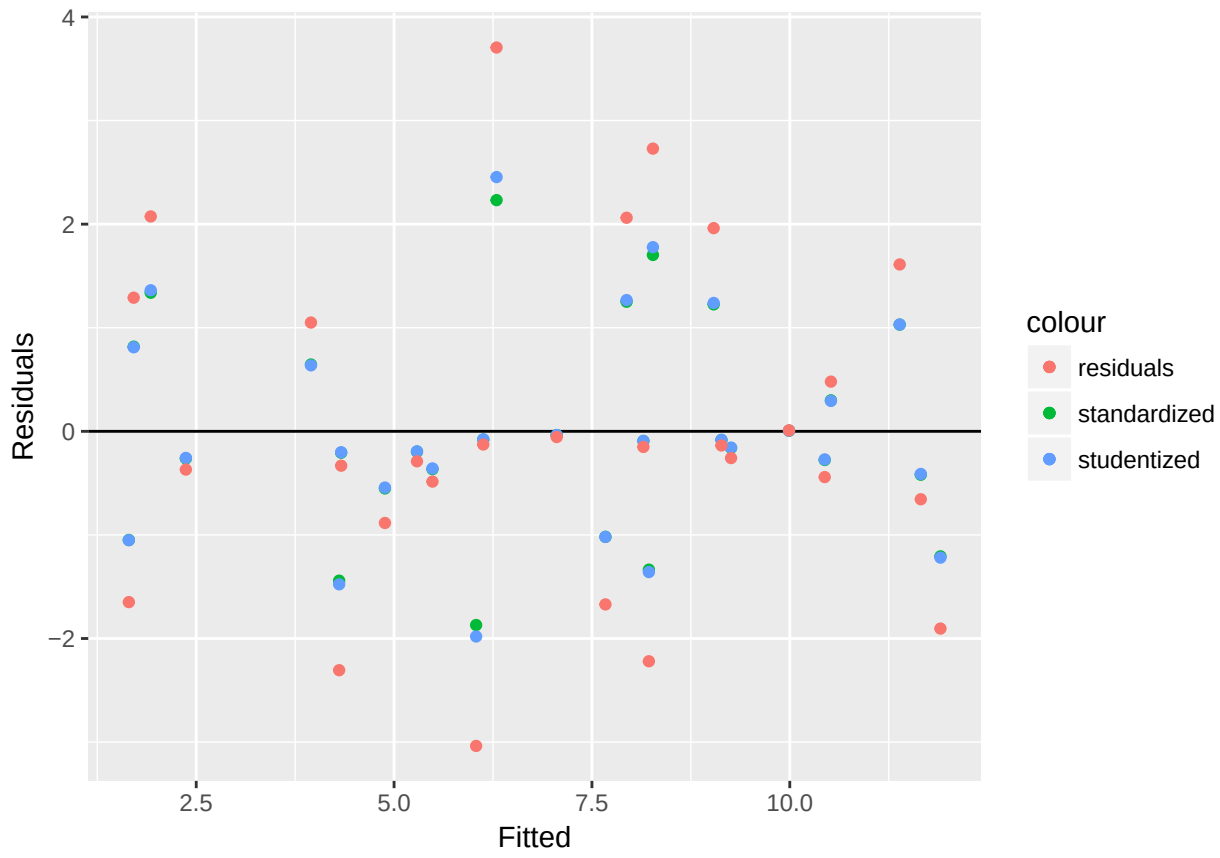

```
plot_d + geom_point(aes(x = x8_residuals))
```



Typically, we plot the residuals to assess whether the model is correctly specified, including checking for linear relationship, constant variance in the residuals, and also for outliers. Whenever there is more than one input variable, this becomes harder to do. Partial regression plots give us a more sophisticated tool for doing this because they allow us to focus our examination of each variable individually. In this case, I would say that heteroskedasticity appears to be present in each of the input variables, whereas earlier, I would say that it was only obvious in x_2 and x_7.

e. Compute the studentized residuals and the R-student residuals for this model. What information is conveyed by these scaled residuals?

```
nfl$rstandard <- rstandard(model)
nfl$rstudent <- rstudent(model)
plot_e <- ggplot(nfl, aes(x = fitted)) + geom_hline(yintercept = 0)+
  geom_point(aes(y = rstandard, color = 'standardized')) +
  geom_point(aes(y = rstudent, color = 'studentized')) +
  geom_point(aes(y = residuals, color = 'residuals')) +
  scale_y_continuous(name = 'Residuals') +
  scale_x_continuous(name = 'Fitted')
plot_e
```



Plotting the residuals helps to identify whether heterskedasticity is present in the data. After scaling the residuals, it is easier to identify whether there are outliers present in the data. In the studentized residuals the numerator (residual) and the denominator (sample variance) are independent of one each other (this is not the case with the standardized residual).

The independence between the numerator and denominator allows us to use the t-distribution (degrees of freedom = n - p - 1) to test the null hypothesis that a given point is an outlier. It is recommended to use the bonferroni correction (alpha / N) since this is a multiple testing problem. Here is a hypothesis test for each of the data points in the dataset:

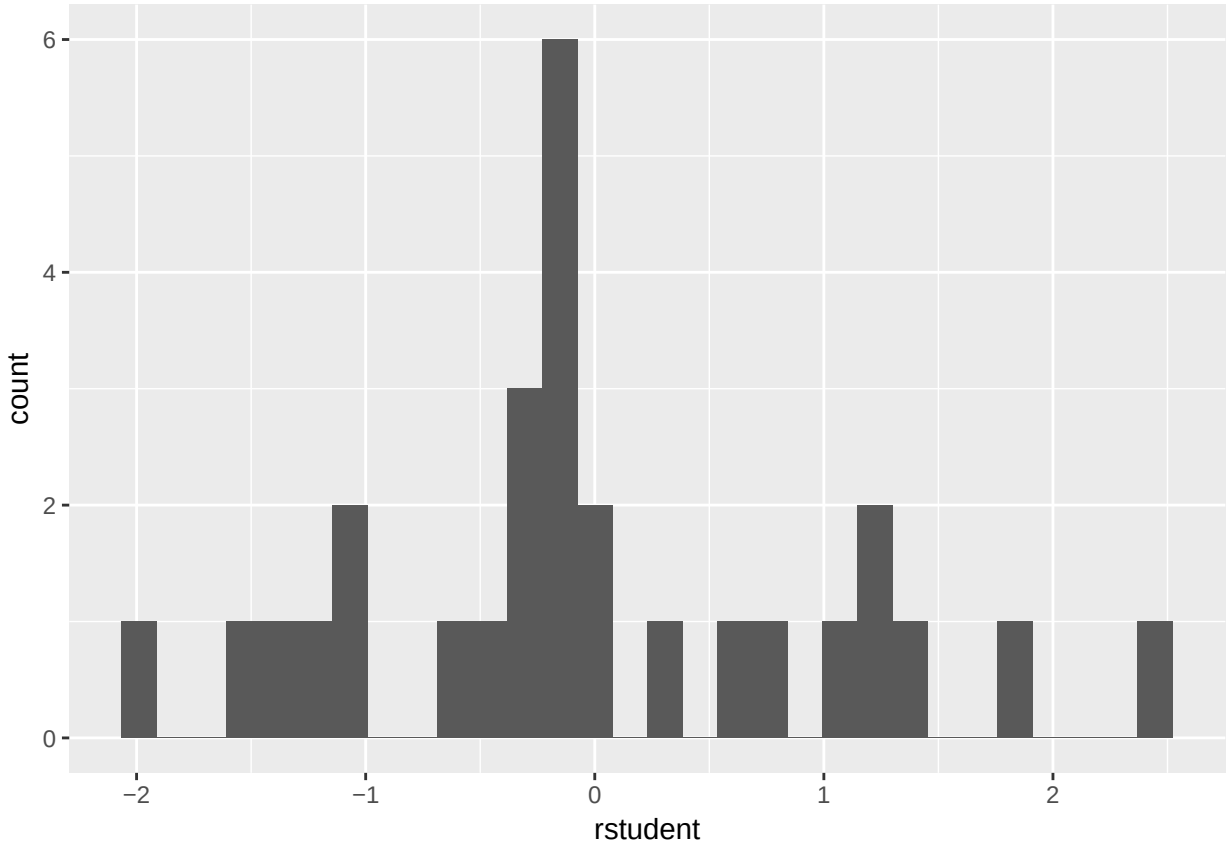
```
t_critical <- qt(1 - .05 / nrow(nfl), df = nrow(nfl) - 4 - 1)
any(abs(nfl$rstudent) > t_critical)
```

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

According to the results of the t-test of studentized residuals with a bonferroni correction ($\alpha/N = 0.00179$), none of the data points are outliers.

In practice, it is commmon to use a histogram to identify outliers as well. Here is a histogram of the studentized residuals.

```
ggplot(nfl, aes(x = rstudent)) + geom_histogram(bins=30)
```

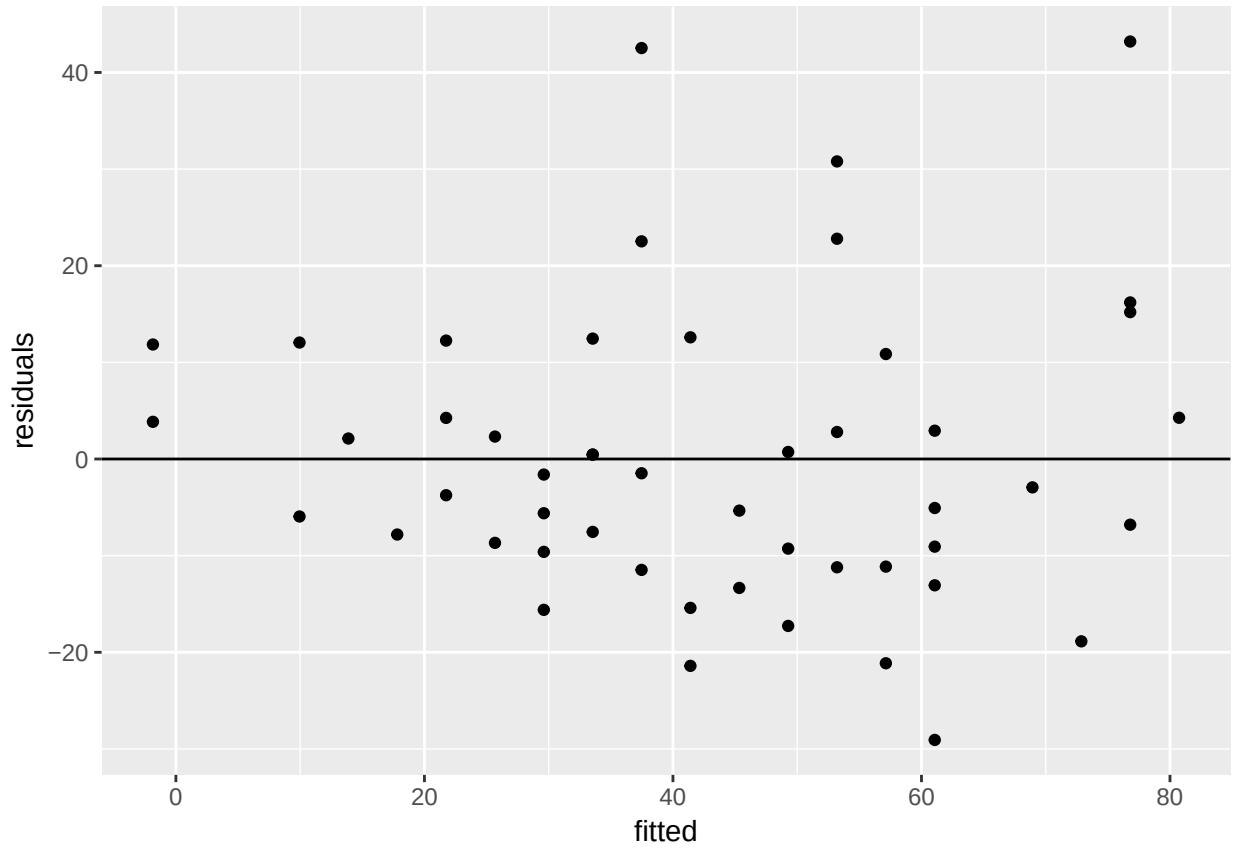


Problem 3

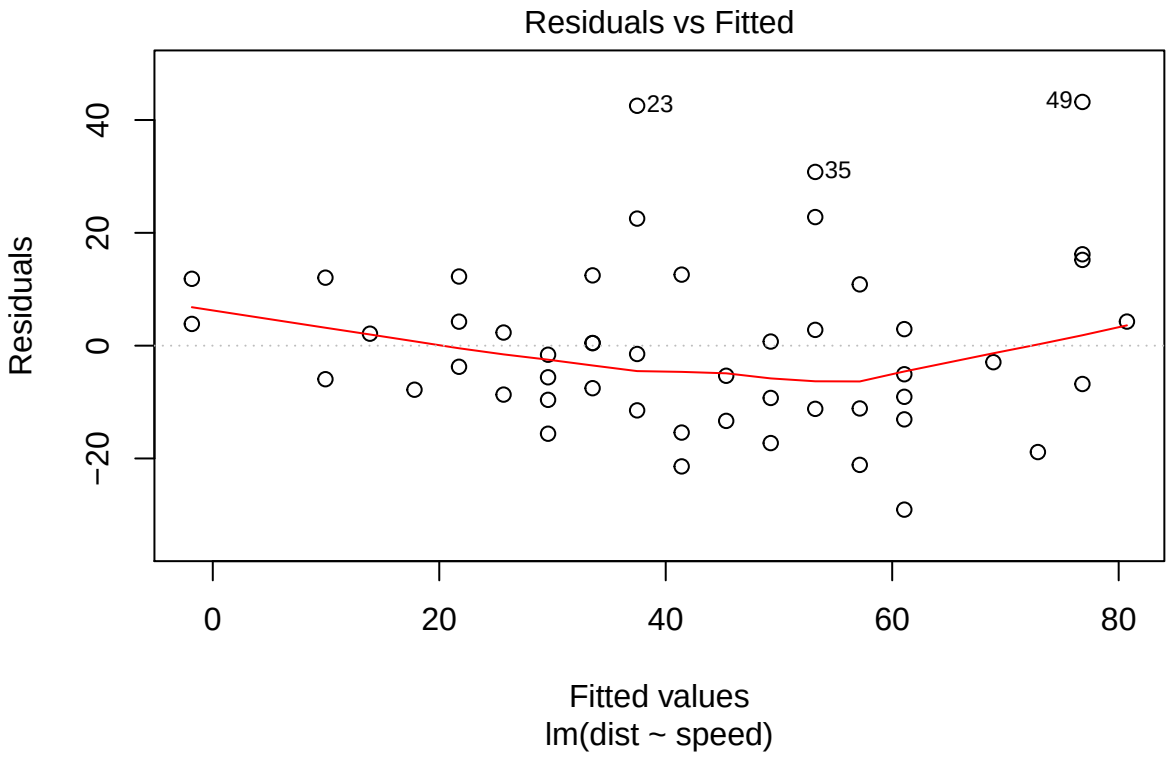
This exercise is designed to show how to check for heteroscedasticity of residuals once you build the linear regression model. Use the cars dataset, which is already stored in R and you can directly type in cars to get access to it.

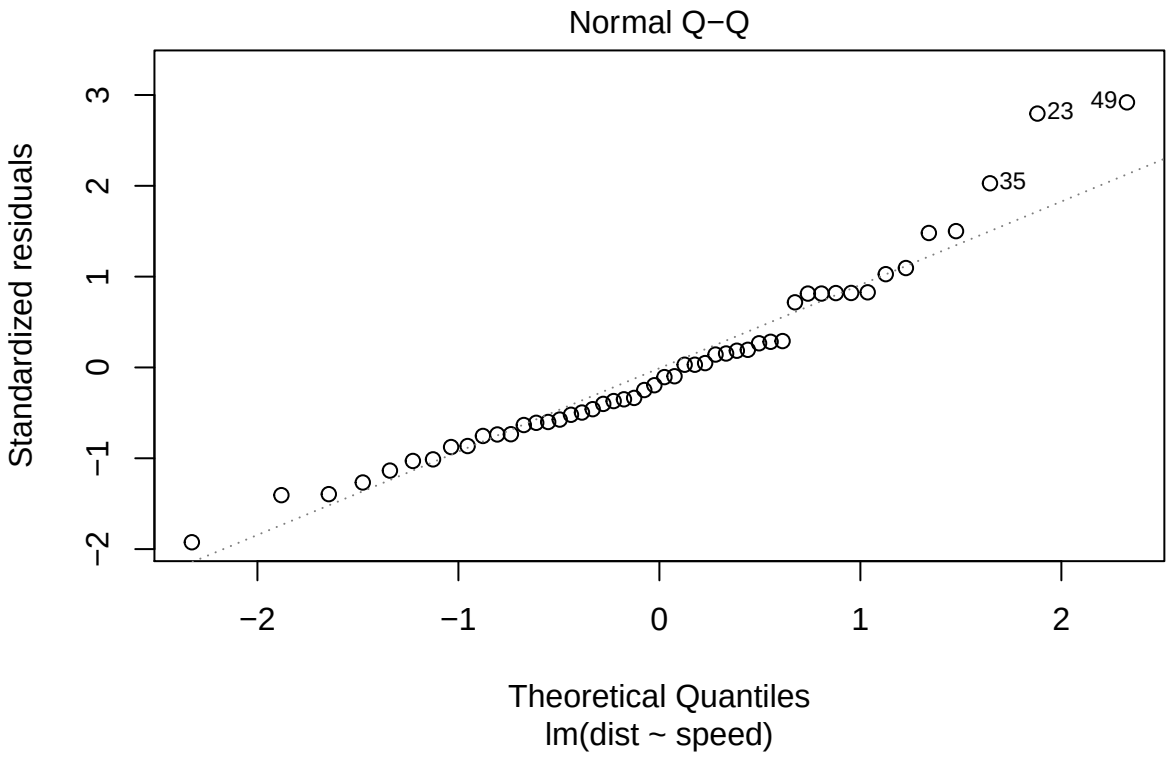
a. Fit dist to speed and store the result object as lm model. Plot the plots of residuals v.s. fitted values and square root of studentized residuals v.s. fitted values by the command plot(lm model). Does heteroscedasticity exist?

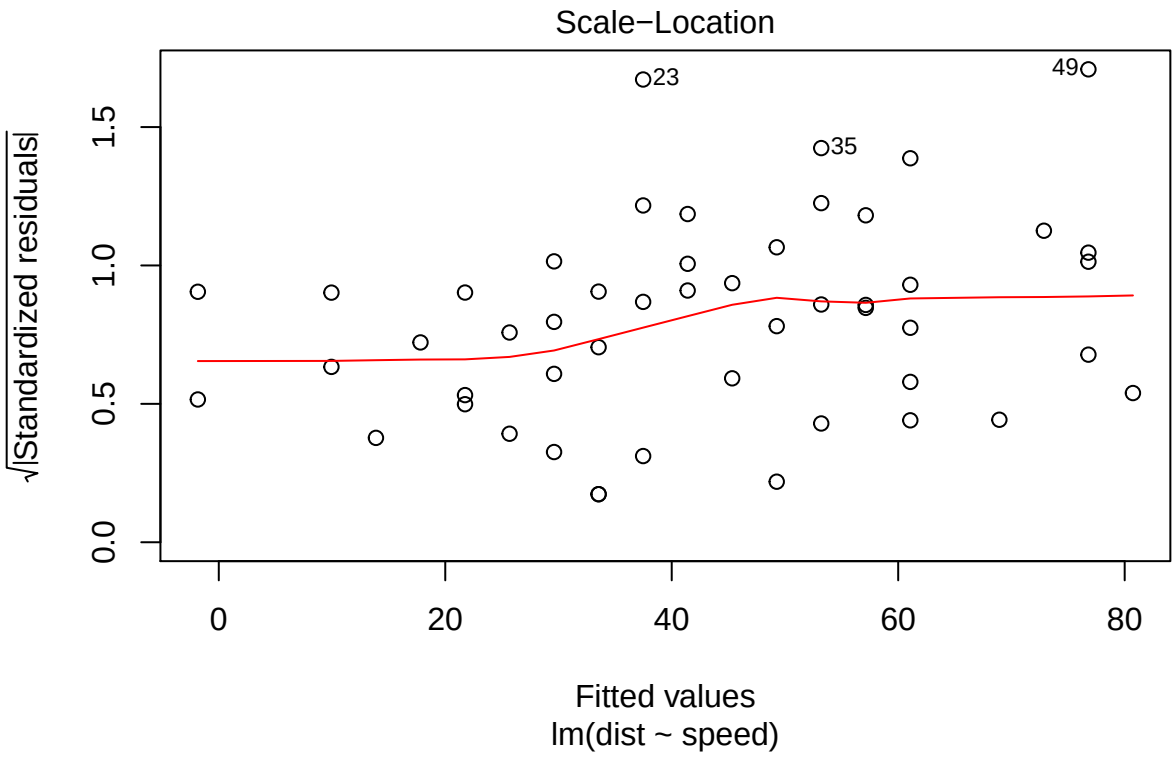
```
model3 <- lm(dist ~ speed, data = cars)
cars$residuals <- model3$residuals
cars$rstudent <- rstudent(model3)
cars$rstudent_sqrt <- sqrt(abs(rstudent(model3)))
cars$fitted <- model3$fitted.values

# Residuals vs. fitted values
ggplot(cars, aes(x = fitted, y = residuals)) + geom_point() +
  geom_hline(yintercept = 0)
```

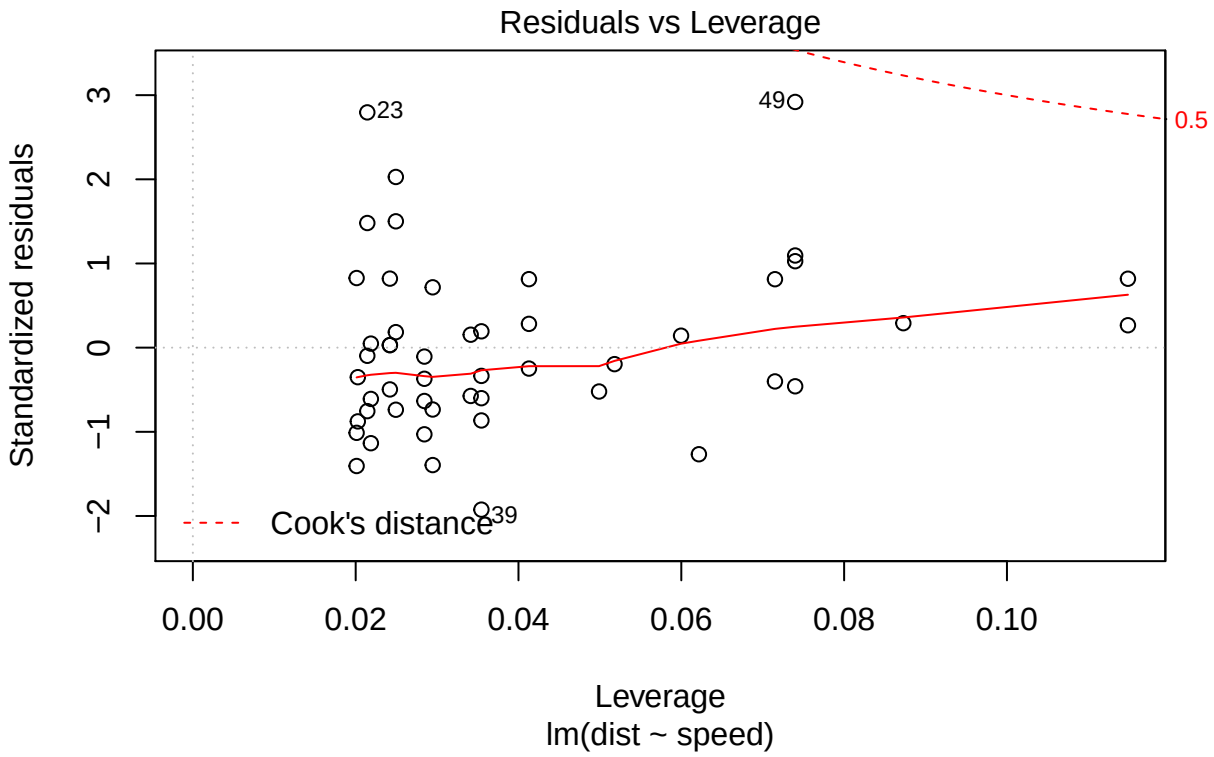



```
plot(model3)
```


Residuals vs Fitted
Residuals
Fitted values
lm(dist ~ speed)
23
35
49
40
20
0
−20
0
20
40
60
80

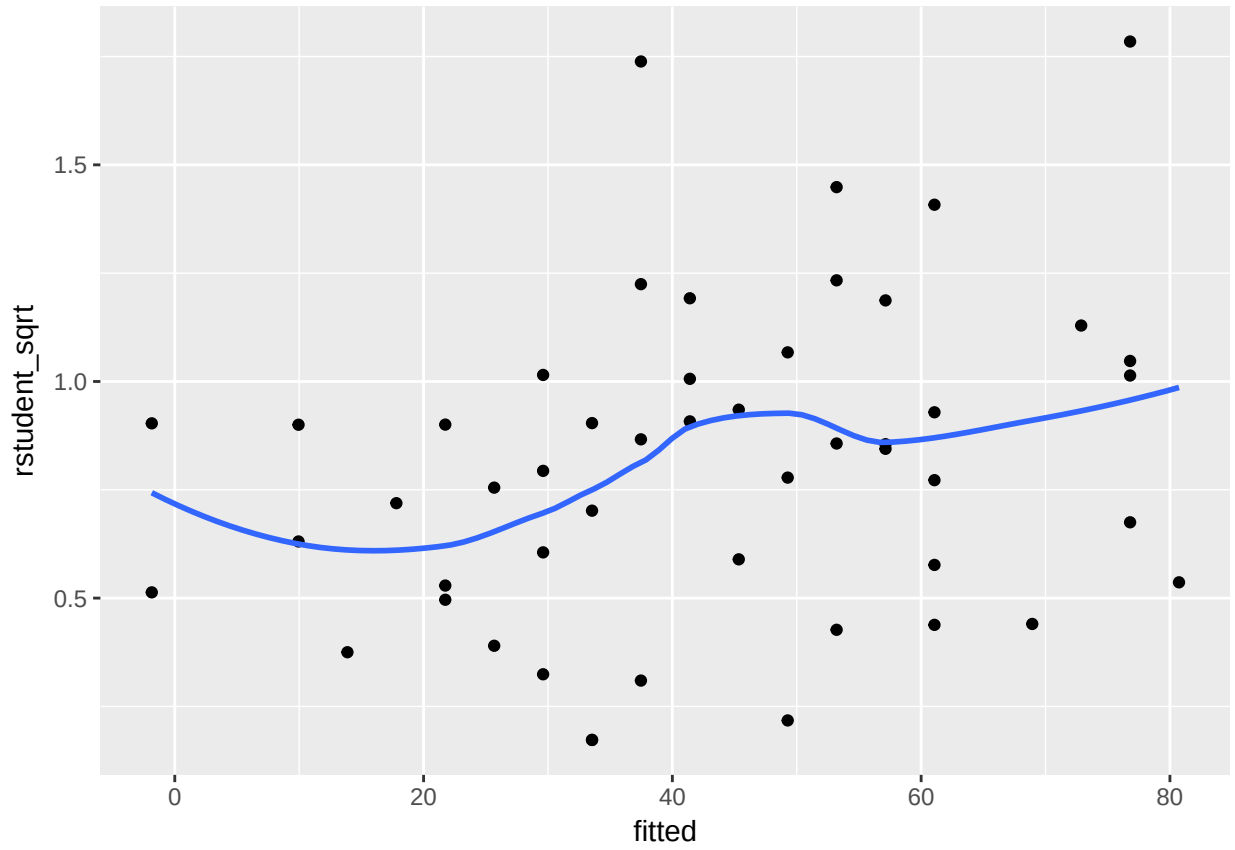

Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(dist ~ speed)
23
49
35
−2
−1
0
1
2
3


Scale−Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
23
35
49
0
20
40
60
80
Fitted values
lm(dist ~ speed)




```
# Square root of studentized residuals vs fitted values
ggplot(cars, aes(x = fitted, y = rstudent_sqrt)) + geom_point() +
  geom_smooth(se=FALSE)
```

```
## `geom_smooth()` using method = 'loess'
```



From this chart it appears that heteroskedasticity exists. From 0 to 20, there is less variation. Then from 20 through 80 there is more variation.

b. Use Breush Pagan Test to check for heteroscedasticity. What's your conclusion?

```
# install.packages('lmtest')
library(lmtest)
```

```
## Warning: package 'lmtest' was built under R version 3.3.3
```

```
## Loading required package: zoo
```

```
## Warning: package 'zoo' was built under R version 3.3.3
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
bp_result <- bptest(model3)
bp_result
```

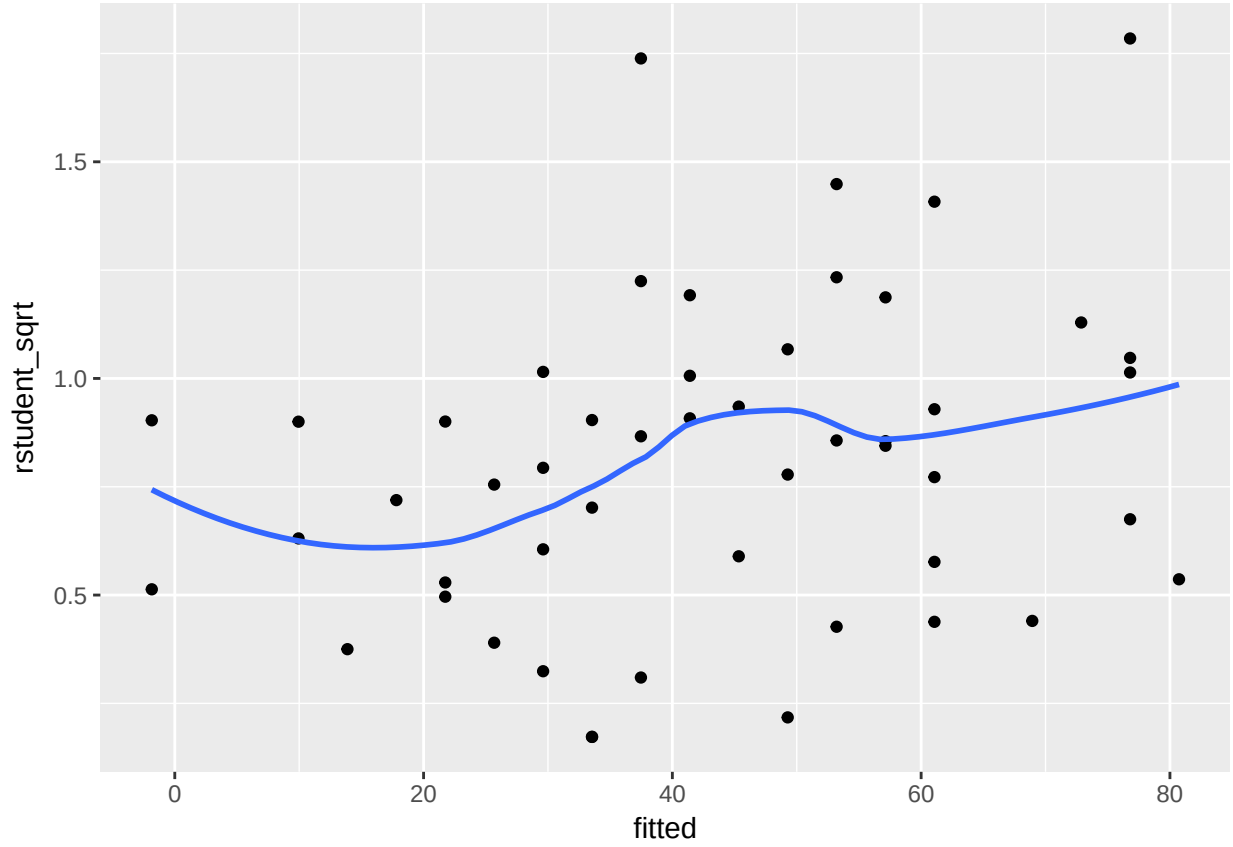
```
##
##  studentized Breusch-Pagan test
##
## data:  model3
## BP = 3.2149, df = 1, p-value = 0.07297
```

According to the Breusch-Pagan test, we do not have evidence of heterskedasticity in the data. The null hypothesis is that the residuals are homoskedastic. The test statistic calculated by the procedure was BP = 3.215 corresponding to a p-value of 0.07297 which is not sufficient to reject the null hypothsis.

c. If there exists heteroscedasticity, use Box-cox transformation rectification. Check the results after the transformation.

```
model3c <- lm(sqrt(dist) ~ speed, data = cars)
cars$rstudent_sqrt_bc <- sqrt(abs(rstudent(model3c)))
# Square root of studentized residuals vs fitted values
ggplot(cars, aes(x = fitted, y = rstudent_sqrt)) + geom_point() +
  geom_smooth(se=FALSE)
```

```
## `geom_smooth()` using method = 'loess'
```




```
bp_result_bc <- bptest(model3c)
bp_result_bc
```

```
##
##  studentized Breusch-Pagan test
##
## data:  model3c
## BP = 0.011192, df = 1, p-value = 0.9157
```

Problem 4

This problem is designed to see the effect of heteroscedasticity on the linear regression model.

Generate 100,000 observations by Eq. (10) for the homoscedastic case and the two types of heteroscedastic cases respectively. Regard the 100,000 observations as the popu- lation, and for N = 25, 50, 100, 250, 500, 1000, uniformaly sample N points from the population without replacement for 1000 replications.

```
# Create a dataframe with all possible combinations of parameters from which to generate the population
var_values <- c(1, 2, 3)
b_values <- c(1, 3, 5)
df_values <- c(1, 2, 3)
p_values <- c(0.2, 0.5, 0.8)
param_comb <- as.data.frame(matrix(nrow = 81, ncol = 4))
colnames(param_comb) <- c('var', 'b', 'df', 'p')
row_num <- 1
for (i in var_values){
  for (j in b_values){
    for (k in df_values){
      for (l in p_values){
        param_comb[row_num,] <- c(var = i, b = j, df = k, p = l)
        row_num <- row_num + 1
      }
    }
  }
}

# Generating the population by sampleing from the distribution combination options and distributions
population <- matrix(nrow = 100000, ncol = 5)

for (i in 1:100000){

  # 1. Select the distribution combination
  i_idx <- sample(1:81, size = 1)
  i_params <- param_comb[i_idx,]

  # 2. Generate a single sample
  population[i, ] <- c(rnorm(n = 1, mean = 0, sd = sqrt(i_params$var))
                       , runif(n = 1, min = -i_params$b, max = i_params$b)
                       , rchisq(n = 1, df = i_params$df)
                       , rbinom(size = 1, n = 1, p = i_params$p)
                       , rchisq(n = 1, df = 5))
}

# Conver to a dataframe
population <- as.data.frame(population)
colnames(population) <- c('x1', 'x2', 'x3', 'x4', 'e')

# Calculate the heteroskedastic errors
population$heter_e1 <- sqrt(population$x3 + 1.6) * population$e
population$heter_e2 <- sqrt(population$x3) * sqrt(population$x4 + 2.5) * population$e

# Define y and 1 vector
population$one <- 1
population$y <- 1 + population$x1 + population$x2 + population$x3 + population$e
population$y2 <- 1 + population$x1 + population$x2 + population$x3 + population$heter_e1
population$y3 <- 1 + population$x1 + population$x2 + population$x3 + population$heter_e2
head(population)
```

```
##           x1         x2        x3 x4        e  heter_e1  heter_e2 one
## 1  1.6030830  0.9543623 1.9239716  1 7.060712 13.254541 18.322378   1
## 2 -1.5774788 -0.8290356 0.2577917  1 2.894688  3.945484  2.749604   1
## 3 -0.5331446 -0.3517195 0.6419924  1 2.322959  3.478233  3.482094   1
## 4  1.2812485 -2.4908284 0.2867359  1 2.727474  3.746417  2.732345   1
## 5 -1.0024292 -1.7318106 0.3912493  1 2.428403  3.426759  2.841722   1
## 6  0.5679879 -3.0258940 3.5573441  1 3.616086  8.212054 12.759560   1
##           y        y2        y3
## 1 12.542129 18.735958 23.803795
## 2  1.745965  2.796761  1.600882
## 3  3.080087  4.235361  4.239223
## 4  2.804630  3.823573  2.809501
## 5  1.085412  2.083769  1.498731
## 6  5.715524 10.311492 14.858998
```

```r
# Create samples of various sizes from the population
get_replications <- function(size = 25){
  lapply(1:1000, function(x){
    return(population[sample(1:100000, size), ])
    })
}

s25 <- get_replications(25)
s50 <- get_replications(50)
s100 <- get_replications(100)
s250 <- get_replications(250)
s500 <- get_replications(500)
s1000 <- get_replications(1000)
```

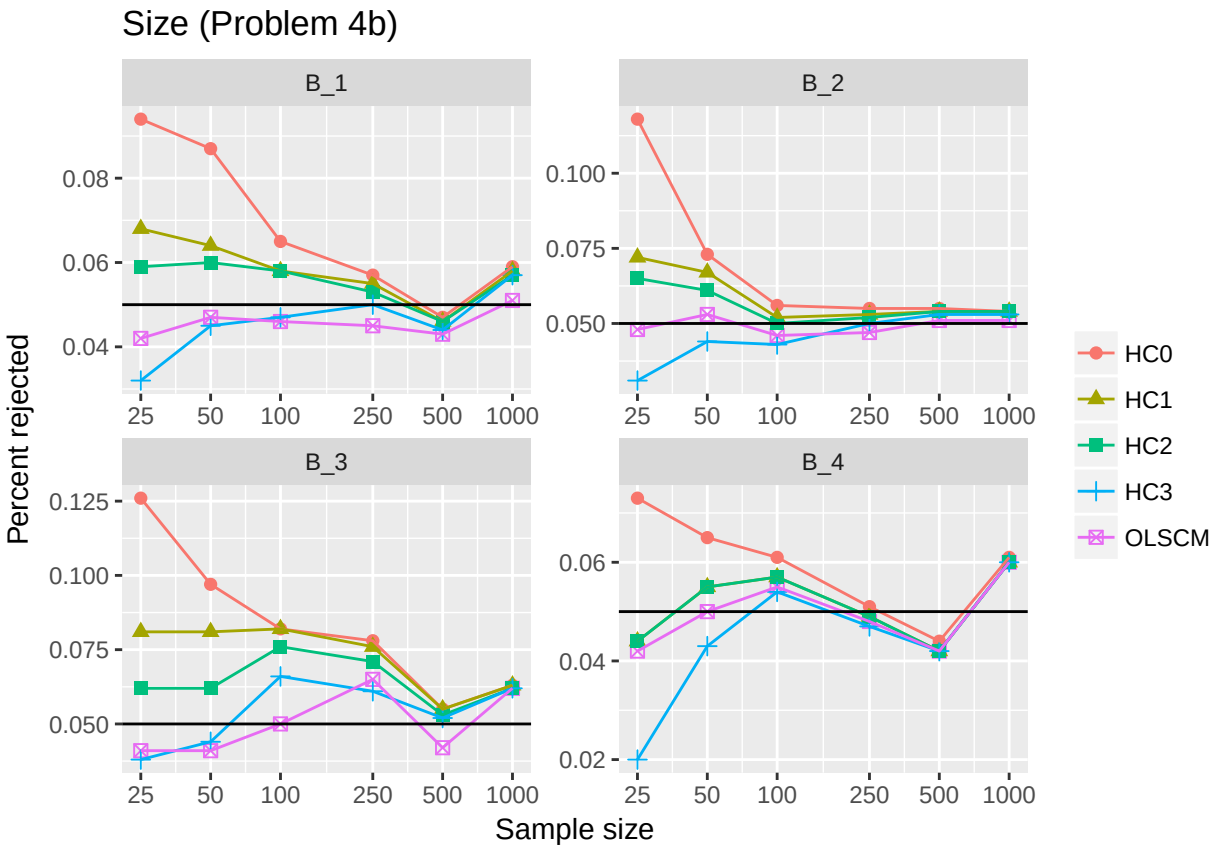
b. Homoscedastic Case. Use the homoscedastic set of data for the following experiment.

```r
calc_beta <- function(X, y){
  # Takes X (n x p) and y (n x 1)
  # Returns:
  #     beta (p x 1)
  X <- as.matrix(X)
  y <- as.matrix(y)
  return(solve(t(X) %*% X) %*% t(X) %*% y)
}

OLSCM <- function(X, e){
  # Takes the design matrix and residual and returns the OLSCM covariance matrix
  e <- as.vector(e)
  sum(e^2)  / (nrow(X) - ncol(X)) * solve(t(X) %*% X)
}

HC0 <- function(X, e){
  # Takes the design matrix and residual and returns the Huber-White
  # covariance matrix
  e <- as.vector(e)
  solve(t(X) %*% X) %*% t(X) %*% diag(e^2) %*% X %*% solve((t(X) %*% X))
}

HC1 <- function(X, e){
  # Takes the design matrix and residual and returns the Huber-White
```

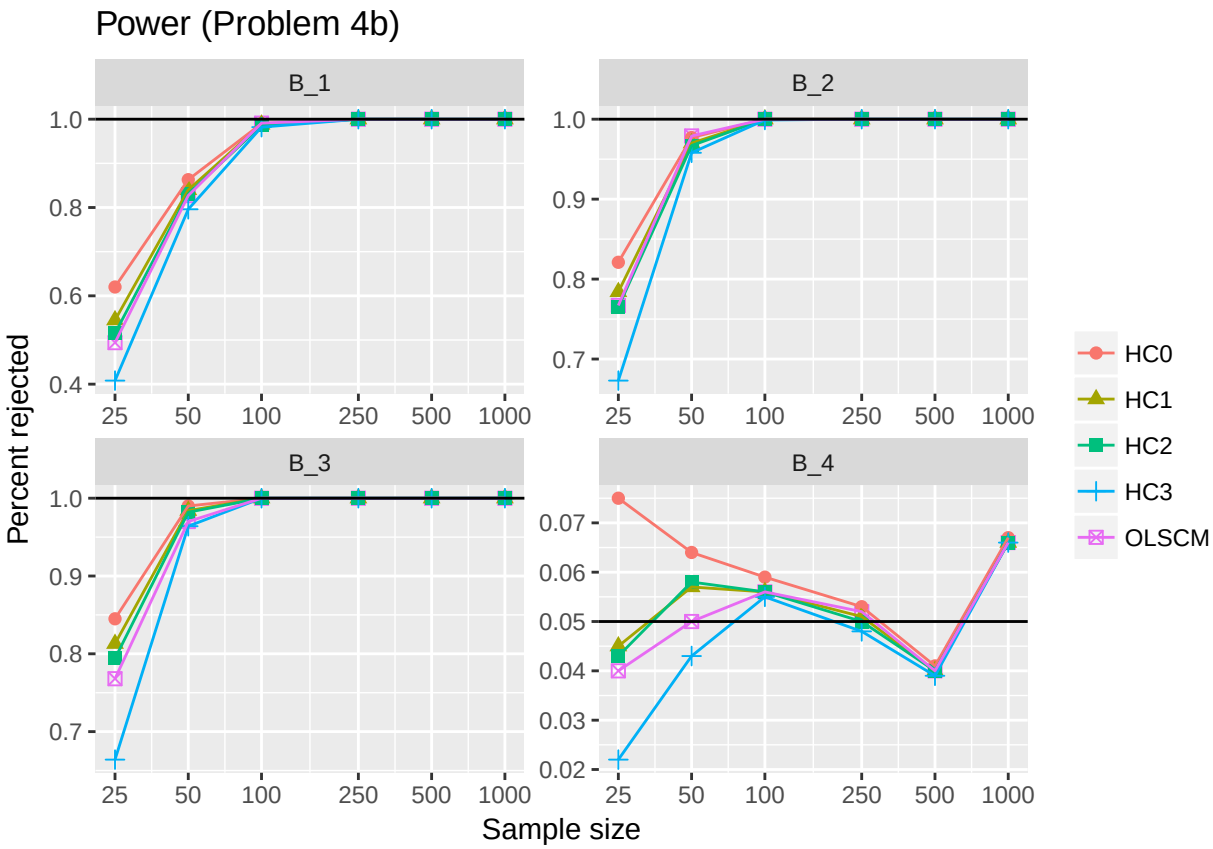
```r
  # covariance matrix
  e <- as.vector(e)
  nrow(X) / (nrow(X) - ncol(X)) * solve(t(X) %*% X) %*% t(X) %*% diag(e^2) %*% X %*% solve(t(X) %*% X)
}

HC2 <- function(X, e){
  # Takes the design matrix and residual and returns the Huber-White
  # covariance matrix
  e <- as.vector(e)
  h_ii <- diag(X %*% solve(t(X) %*% X) %*% t(X))
  solve(t(X) %*% X) %*% t(X) %*% diag(e^2 / (1 - h_ii))  %*% X %*% solve(t(X) %*% X)
}

HC3 <- function(X, e){
  # Takes the design matrix and residual and returns the Huber-White
  # covariance matrix
  e <- as.vector(e)
  h_ii <- diag(X %*% solve(t(X) %*% X) %*% t(X))
  solve(t(X) %*% X) %*% t(X) %*% diag(e^2 / (1 - h_ii)^2)  %*% X %*% solve(t(X) %*% X)
}

simulate_test <- function(sample_list, true_beta, n, y = 'y'){
  # Take a sample of data test null hypothesis that beta is equal
  # to population beta values
  # Returns
  #    Proportion of times we reject the null hypothesis
  num_replications <- 1000

  t_olscm <- matrix(nrow = num_replications, ncol = 5)
  t_hc0 <- matrix(nrow = num_replications, ncol = 5)
  t_hc1 <- matrix(nrow = num_replications, ncol = 5)
  t_hc2 <- matrix(nrow = num_replications, ncol = 5)
  t_hc3 <- matrix(nrow = num_replications, ncol = 5)

  t_0_olscm <- matrix(nrow = num_replications, ncol = 5)
  t_0_hc0 <- matrix(nrow = num_replications, ncol = 5)
  t_0_hc1 <- matrix(nrow = num_replications, ncol = 5)
  t_0_hc2 <- matrix(nrow = num_replications, ncol = 5)
  t_0_hc3 <- matrix(nrow = num_replications, ncol = 5)

  for(i in 1:num_replications){
    sample_i <- sample_list[[i]]
    X_i <- as.matrix(sample_i[,c('one', 'x1', 'x2', 'x3', 'x4')])
    y_i <- as.matrix(sample_i[,y])
    beta_sample <- calc_beta(X_i, y_i)
    e_i <- X_i %*% beta_sample - y_i
    var_beta_olscm <- OLSCM(X_i, e_i)
    var_beta_hc0 <- HC0(X_i, e_i)
    var_beta_hc1 <- HC1(X_i, e_i)
    var_beta_hc2 <- HC2(X_i, e_i)
    var_beta_hc3 <- HC3(X_i, e_i)

    # H_o: Beta = Beta* (for Size calculation)
```

```
    t_olscm[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_olscm)))
    t_hc0[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_hc0)))
    t_hc1[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_hc1)))
    t_hc2[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_hc2)))
    t_hc3[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_hc3)))

    # H_o: Beta = 0 (for Power calculations)
    t_0_olscm[i,] <-  t((beta_sample) / sqrt(diag(var_beta_olscm)))
    t_0_hc0[i,] <-  t((beta_sample) / sqrt(diag(var_beta_hc0)))
    t_0_hc1[i,] <-  t((beta_sample) / sqrt(diag(var_beta_hc1)))
    t_0_hc2[i,] <-  t((beta_sample) / sqrt(diag(var_beta_hc2)))
    t_0_hc3[i,] <-  t((beta_sample) / sqrt(diag(var_beta_hc3)))
  }

  prop_reject <- matrix(nrow = 10, ncol = 5)

  # What proportion are we incorrectly rejecting (H_o: Beta = Beta*)?
  prop_reject[1,] <- apply(t_olscm, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000}
  prop_reject[2,] <- apply(t_hc0, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000})
  prop_reject[3,] <- apply(t_hc1, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000})
  prop_reject[4,] <- apply(t_hc2, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000})
  prop_reject[5,] <- apply(t_hc3, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000})

  # What proportion are we correctly rejecting (H_o: Beta = 0)?
  prop_reject[6,] <- apply(t_0_olscm, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 100
  prop_reject[7,] <- apply(t_0_hc0, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000}
  prop_reject[8,] <- apply(t_0_hc1, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000}
  prop_reject[9,] <- apply(t_0_hc2, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000}
  prop_reject[10,] <- apply(t_0_hc3, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000

  prop_reject
}

format_df <- function(m, n){
  df <- as.data.frame(m)
  colnames(df) <- c('B_0', 'B_1', 'B_2', 'B_3', 'B_4')
  df$se <- c('OLSCM', 'HC0', 'HC1', 'HC2', 'HC3')
  df$test <- c(rep('Size', 5), rep('Power', 5))
  df$n <- n
  df
}

beta_pop1 <- calc_beta(population[,c('one', 'x1', 'x2', 'x3', 'x4')], population$y)

prop_reject <- format_df(simulate_test(s25, beta_pop1, n = 25), n = 25)
prop_reject <- rbind(prop_reject, format_df(simulate_test(s50, beta_pop1, n = 50), n = 50))
prop_reject <- rbind(prop_reject, format_df(simulate_test(s100, beta_pop1, n = 100), n = 100))
prop_reject <- rbind(prop_reject, format_df(simulate_test(s250, beta_pop1, n = 250), n = 250))
prop_reject <- rbind(prop_reject, format_df(simulate_test(s500, beta_pop1, n = 500), n = 500))
prop_reject <- rbind(prop_reject, format_df(simulate_test(s1000, beta_pop1, n = 1000), n = 1000))

prop_reject_long<- melt(prop_reject[,-1], id.vars = c('se', 'test', 'n'))
```

```
plot_power_size <- function(df, test = 'Size', title){
  ggplot(df[df$test == test,], aes(x = n, y = value, color = se)) +
  geom_point(aes(shape = se), size = 2) +
  geom_line() +
  scale_x_continuous(breaks = c(25, 50, 100, 250, 500, 1000), trans = 'log') +
  facet_wrap(~variable, scales = 'free', nrow = 2, labeller = label_wrap_gen(multi_line=FALSE)) +
  labs(x = 'Sample size'
       , y = 'Percent rejected'
       , title = title) +
  theme(legend.title = element_blank())
}

plot_power_size(prop_reject_long, 'Size', 'Size (Problem 4b)') +
  geom_hline(yintercept = 0.05)
```




```
plot_power_size(prop_reject_long, 'Power', 'Power (Problem 4b)') +
    geom_hline(aes(yintercept = ifelse(prop_reject_long[prop_reject_long$test == 'Power',]$variable ==
```



Size Properties

Size = Null hypotheis is true. Probability of incorrectly rejecting null hypothesis

- Tests using the the OLSCM coviance estimator has good size properties.
- Tests using the HC0 and HC1 covariance estimator does not have good size properties for small samples
- Tests using the HC2, HC3 can have good size properties but they are not consistently good for small samples. For example, HC2 has bad size properties for small samples for B_2 and B_3.
- For large sample sizes, all estimators appear to have similar size properties.

Power Properties

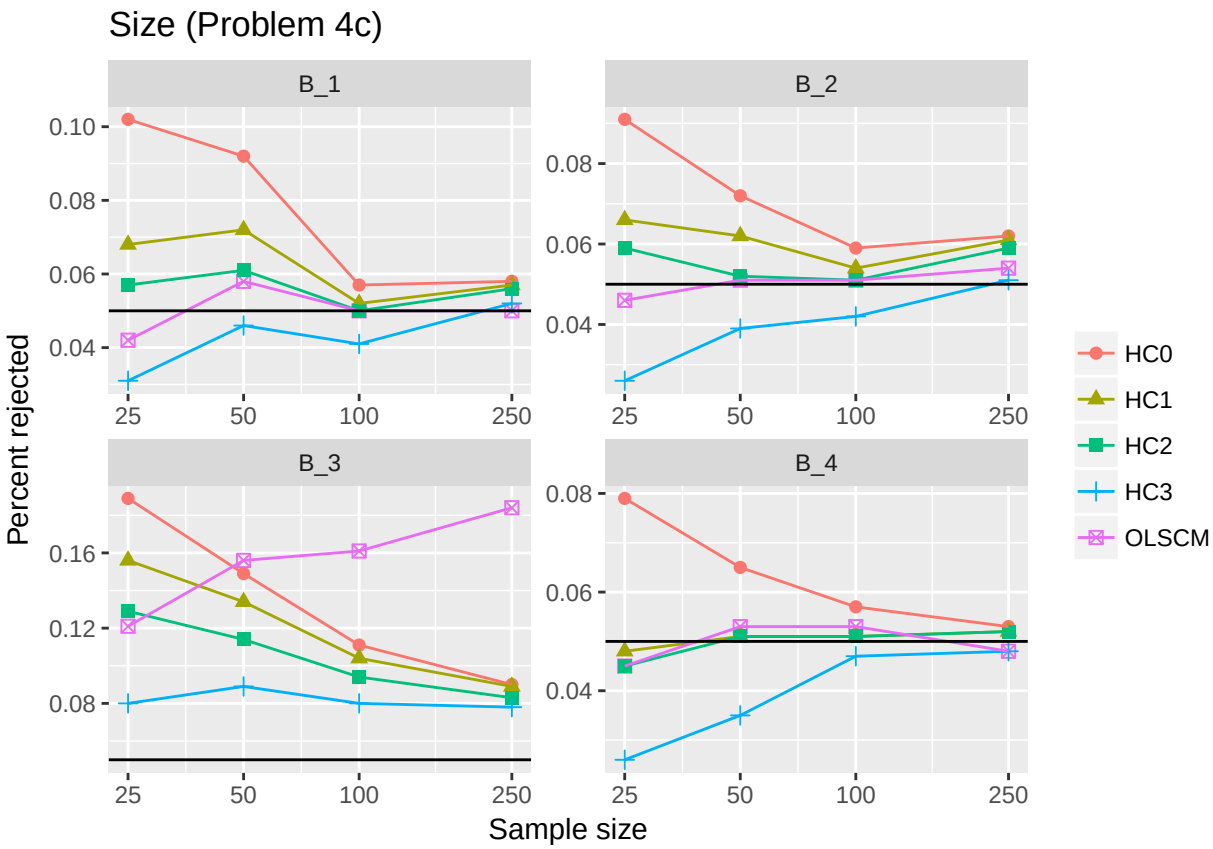
Power = Null hypothesis is false. Probability of correctly rejecting null hypothesis

- Tests for B_1 through B_3 have the highest power, but it also has the highest probability of falsely rejecting the null hypothesis.

c. Heteroscedastic Case. Using the first type of heteroscedastic set of data, draw the four plots for empirical size and four plots for empirical power similarly to part b above. Which covariance estimator is favarable in this setup. And what can you conclude?

```
beta_pop2 <- calc_beta(population[,c('one', 'x1', 'x2', 'x3', 'x4')], population$y2)

prop_rejectc <- format_df(simulate_test(s25, beta_pop2, n = 25, 'y2'), n = 25)
prop_rejectc <- rbind(prop_rejectc, format_df(simulate_test(s50, beta_pop2, n = 50, 'y2'), n = 50))
prop_rejectc <- rbind(prop_rejectc, format_df(simulate_test(s100, beta_pop2, n = 100, 'y2'), n = 100))
prop_rejectc <- rbind(prop_rejectc, format_df(simulate_test(s250, beta_pop2, n = 250, 'y2'), n = 250))
# prop_rejectc <- rbind(prop_rejectc, format_df(simulate_test(s500, beta_pop2, n = 500, 'y2'), n = 500))
# prop_rejectc <- rbind(prop_rejectc, format_df(simulate_test(s1000, beta_pop2, n = 1000, 'y2'), n = 10
```

```
prop_reject_longc<- melt(prop_rejectc[,-1], id.vars = c('se', 'test', 'n'))

plot_power_size(prop_reject_longc, 'Size', 'Size (Problem 4c)') +
  geom_hline(yintercept = 0.05)
```



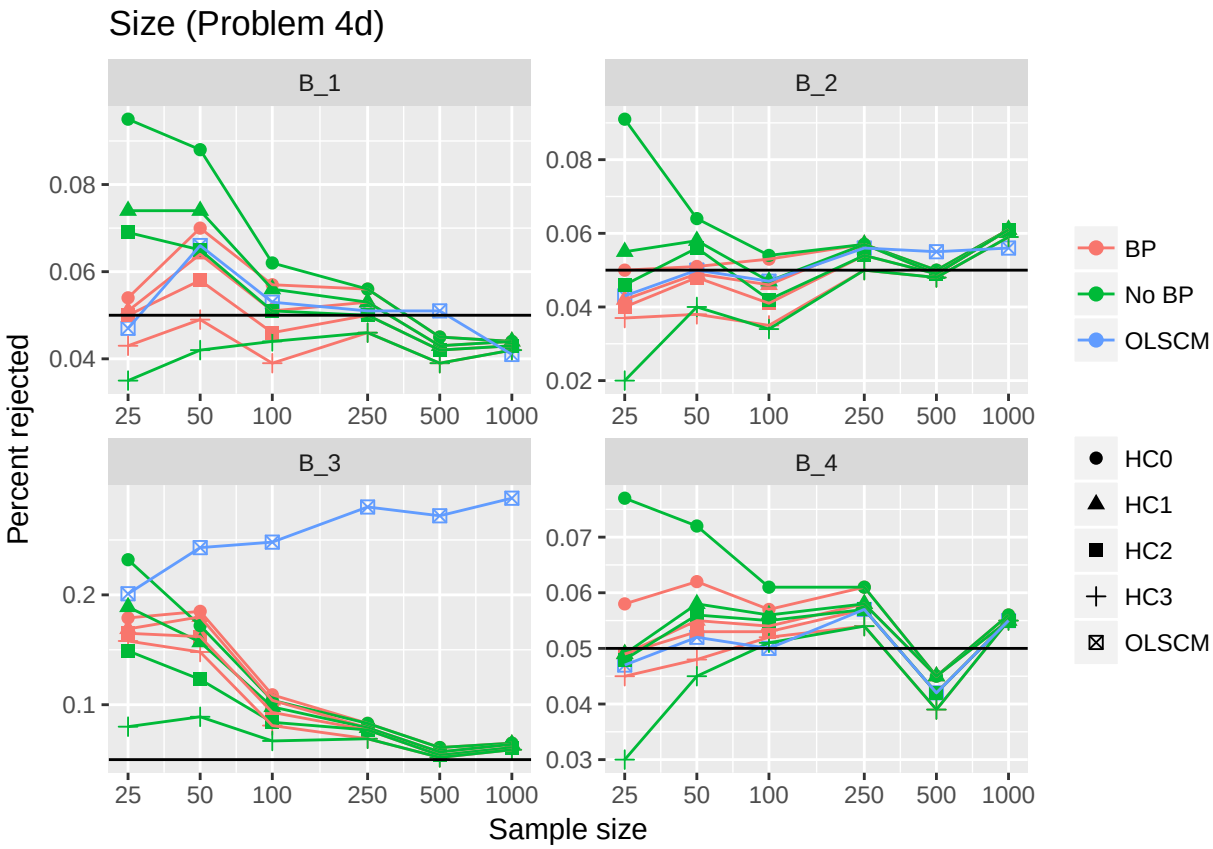
*d. Screening for Heteroscedasticity. We propose a new procedure to tackle the heteroscedasticity. * Step 1 Use Breusch and Pagan (BP) test to determine if there is heteroscedasticity. * Step 2 If there is no heteroscedasticity, apply OLSCM, otherwise apply one variant of HC's.*

Now, we have 9 methods, i.e., standard OLSCM test, HCm test regardless of the results of Breusch and Pagan test, m = 0,1,2,3, and HCm test with BP test, m = 0,1,2,3. Using the second type of heteroscedastic set of data, only draw the plots for empirical size. Is the BP procedure better than others? What can you conclude?

```
simulate_test_d <- function(sample_list, true_beta, n){
  # Take a sample of data test null hypothesis that beta is equal
  # to population beta values
  # Returns
  #    Proportion of times we reject the null hypothesis
  num_replications <- 1000

  t_olscm <- matrix(nrow = num_replications, ncol = 5)

  t_hc0_bp <- matrix(nrow = num_replications, ncol = 5)
  t_hc1_bp <- matrix(nrow = num_replications, ncol = 5)
  t_hc2_bp <- matrix(nrow = num_replications, ncol = 5)
  t_hc3_bp <- matrix(nrow = num_replications, ncol = 5)
```

```r
t_hc0 <- matrix(nrow = num_replications, ncol = 5)
t_hc1 <- matrix(nrow = num_replications, ncol = 5)
t_hc2 <- matrix(nrow = num_replications, ncol = 5)
t_hc3 <- matrix(nrow = num_replications, ncol = 5)

for(i in 1:num_replications){
  sample_i <- sample_list[[i]]
  X_i <- as.matrix(sample_i[,c('one', 'x1', 'x2', 'x3', 'x4')])
  y_i <- as.matrix(sample_i$y3)
  heteroskedasticity <- bptest(y3 ~ x1 + x2 + x3 + x4, data = sample_i)$p.value < 0.05
  beta_sample <- calc_beta(X_i, y_i)
  e_i <- X_i %*% beta_sample - y_i
  var_beta_olscm <- OLSCM(X_i, e_i)
  var_beta_hc0 <- HC0(X_i, e_i)
  var_beta_hc1 <- HC1(X_i, e_i)
  var_beta_hc2 <- HC2(X_i, e_i)
  var_beta_hc3 <- HC3(X_i, e_i)

  # H_o: Beta = Beta*
  t_olscm[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_olscm)))

  # If (heteroskedasticity present)
  # Then test using HC0, HC1, HC2, HC3
  # Else test using OLSCM
  if(heteroskedasticity){
    t_hc0_bp[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_hc0)))
    t_hc1_bp[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_hc1)))
    t_hc2_bp[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_hc2)))
    t_hc3_bp[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_hc3)))
  }else{
    t_hc0_bp[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_olscm)))
    t_hc1_bp[i,] <-  t_hc0_bp[i,]
    t_hc2_bp[i,] <-  t_hc0_bp[i,]
    t_hc3_bp[i,] <-  t_hc0_bp[i,]
  }

  # Same as before
  t_hc0[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_hc0)))
  t_hc1[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_hc1)))
  t_hc2[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_hc2)))
  t_hc3[i,] <-  t((beta_sample - true_beta) / sqrt(diag(var_beta_hc3)))
}

prop_reject <- matrix(nrow = 9, ncol = 5)

# What proportion are we incorrectly rejecting (H_o: Beta = Beta*)?
prop_reject[1,] <- apply(t_olscm, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000}

# With BP test
prop_reject[2,] <- apply(t_hc0_bp, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000
prop_reject[3,] <- apply(t_hc1_bp, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000
prop_reject[4,] <- apply(t_hc2_bp, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000
prop_reject[5,] <- apply(t_hc3_bp, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000
```

```
  # No BP test
  prop_reject[6,] <- apply(t_hc0, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000})
  prop_reject[7,] <- apply(t_hc1, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000})
  prop_reject[8,] <- apply(t_hc2, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000})
  prop_reject[9,] <- apply(t_hc3, 2, function(x) {sum(abs(x) > qt(p = 1 - 0.025, df = n - 5)) / 1000})

  prop_reject
}

format_df_d <- function(m, n){
  df <- as.data.frame(m)
  colnames(df) <- c('B_0', 'B_1', 'B_2', 'B_3', 'B_4')
  df$bp <- c('OLSCM', rep('BP', 4), rep('No BP', 4))
  df$hc <- c(NA, as.character(rep(0:3, 2)))
  df$se <- c('OLSCM', rep(c('HC0', 'HC1', 'HC2', 'HC3'), 2))
  df$test <- c(rep('Size', 9))
  df$n <- n
  df
}

beta_pop3 <- calc_beta(population[,c('one', 'x1', 'x2', 'x3', 'x4')], population$y3)
prop_rejectd <- format_df_d(simulate_test_d(s25, beta_pop3, n = 25), n = 25)
prop_rejectd <- rbind(prop_rejectd, format_df_d(simulate_test_d(s50, beta_pop3, n = 50), n = 50))
prop_rejectd <- rbind(prop_rejectd, format_df_d(simulate_test_d(s100, beta_pop3, n = 100), n = 100))
prop_rejectd <- rbind(prop_rejectd, format_df_d(simulate_test_d(s250, beta_pop3, n = 250), n = 250))
prop_rejectd <- rbind(prop_rejectd, format_df_d(simulate_test_d(s500, beta_pop3, n = 500), n = 500))
prop_rejectd <- rbind(prop_rejectd, format_df_d(simulate_test_d(s1000, beta_pop3, n = 1000), n = 1000))

prop_reject_longd <- melt(prop_rejectd[,-1], id.vars = c('bp', 'hc', 'se', 'test', 'n'))

# Dashed lines for BP, same colors and symbols though


plot_4d <- function(df, test = 'Size', title){
  ggplot(df, aes(x = n, y = value, colour = bp)) +
  geom_point(aes(shape = se), size = 2) +
  geom_line(aes(group = paste(se, bp))) +
  scale_x_continuous(breaks = c(25, 50, 100, 250, 500, 1000), trans = 'log') +
  facet_wrap(~variable, scales = 'free', nrow = 2, labeller = label_wrap_gen(multi_line=FALSE)) +
  labs(x = 'Sample size'
       , y = 'Percent rejected'
       , title = title) +
  theme(legend.title = element_blank())  +
  geom_hline(yintercept = 0.05)
}

plot_4d(prop_reject_longd, 'Size', 'Size (Problem 4d)')
```



Problem 5

1. First, write code to evaluate the denominator using apply or lapply or sapply. Make sure to calcualte all the terms in f on the log scale to avoid numerical issues, before exponentiating and summing. Describe briefly what happes if you do not do the calculation on the log scale.

```
norm_const_log_scale <- function(n, k, p = 0.3){
  if(n == k){
    # Using log[(n-k)^(n-k)] when n == k because log(0) = -Inf
    lchoose(n, k) + 1/2 * k * log(k) + 1/2 * log((n - k)^(n - k)) - 1/2 * n * log(n) + k * 1/2 * log(p)
  }else if(k == 0){
    # Using (n-k)*log[n-k] all other times because log(2000^2000) = Inf
    lchoose(n, k) + 1/2 * log(k^k) + 1/2 * (n - k) * log(n - k) - 1/2 * n * log(n) + k * 1/2 * log(p)
  }else{
    # Using (n-k)*log[n-k] all other times because log(2000^2000) = Inf
    lchoose(n, k) + 1/2 * k * log(k) + 1/2 * (n - k) * log(n - k) - 1/2 * n * log(n) + k * 1/2 * log(p)
  }
}

total_dispersion <- function(n){
  sum(exp(sapply(0:n, function(i) {norm_const_log_scale(n = n, k = i)})))
}

total_dispersion(2000)
```

```
## [1] 1.241513e+114
```

```
# At what point does R treat (n - k)^(n - k) to be infinity? k = 1856
# for (i in seq(2000, 1, by = -1)){
#   result <- (2000 - i) ^ (2000 - i)
#   if(is.infinite(result)) stop('k = ', i)
# }
```

If we do not use the log scale to compute this problem, we will run into issues for large values of $n - k$. e.g., when $n = 2000$, and for k values of 1856 or smaller, $\frac{(n-k)^{n-k}}{n^n}$ will be evaluated by R as `Inf/Inf = NaN`.

Now write code to do the calculation in a fully vectorized fashion with no loops or apply() functions. Using the timing function benchmark() in the rbenchmark package, compare the relative timing (a) and (b) for n = 100,500,1000,2000. Note that for benchmark(), you need multiple replications (100 or 1000) in order to obtain a robust timing.

```
# 1. Vectorizing with an if statement
norm_const_log_scale_vect_1 <- function(n, k, p = 0.3){
  ifelse(k == n, lchoose(n, k) + 1/2 * k * log(k) + 1/2 * log((n - k)^(n - k)) - 1/2 * n * log(n) + k *
         , ifelse(k == 0, lchoose(n, k) + 1/2 * log(k^k) + 1/2 * (n - k) * log(n - k) - 1/2 * n * log(n
                  lchoose(n, k) + 1/2 * k * log(k) + 1/2 * (n - k) * log(n - k) - 1/2 * n * log(n) + k
}

total_dispersion_vect_1 <- function(n){
  sum(exp(norm_const_log_scale_vect_1(n, 0:n)))
}

# 2. Vectorizing with a which statement
norm_const_log_scale_vect_2 <- function(n, k, p = 0.3){
  k_n <- k[which(k == n)]
  k_o <- k[which(k == 0)]
  k_else <- k[which((k != 0) & (k != n))]
  c(lchoose(n, k_n) + 1/2 * k_n * log(k_n) + 1/2 * log((n - k_n)^(n - k_n)) - 1/2 * n * log(n) + k_n *
    ,  lchoose(n, k_o) + 1/2 * log(k_o^k_o) + 1/2 * (n - k_o) * log(n - k_o) - 1/2 * n * log(n) + k_o *
    ,  lchoose(n, k_else) + 1/2 * k_else * log(k_else) + 1/2 * (n - k_else) * log(n - k_else) - 1/2 * n
}

total_dispersion_vect_2 <- function(n){
  sum(exp(norm_const_log_scale_vect_2(n, 0:n)))
}

# Test that we are getting the same results
total_dispersion_vect_1(200) == total_dispersion(200)
```

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

```
total_dispersion_vect_1(2000) == total_dispersion(2000)
```

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

```
total_dispersion_vect_2(200) == total_dispersion(200)
```

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

```
total_dispersion_vect_2(2000) == total_dispersion(2000)
```

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

```
# Conduct benchmarking
library(rbenchmark)
```

```
## Warning: package 'rbenchmark' was built under R version 3.3.2
```

```
get_benchmark_results <- function(n = 2000, r = 100){
  df <- benchmark(replications=c(r),
          original=total_dispersion(n),
          vectorized_1=total_dispersion_vect_1(n),
          vectorized_2=total_dispersion_vect_2(n),
          columns=c('test', 'elapsed', 'replications'))
  df <- dcast(df,  n ~ test, value.var = 'elapsed')
  df$speedup_1 <- df$original / df$vectorized_1
  df$speedup_2 <- df$original / df$vectorized_2
  df$n <- n
  df
}

results <- get_benchmark_results(100)
results <- rbind(results, get_benchmark_results(500))
results <- rbind(results, get_benchmark_results(1000))
results <- rbind(results, get_benchmark_results(2000))
results
```

```
##      n original vectorized_1 vectorized_2 speedup_1 speedup_2
## 1  100     0.11         0.02         0.01  5.500000  11.00000
## 2  500     0.50         0.11         0.03  4.545455  16.66667
## 3 1000     1.03         0.21         0.07  4.904762  14.71429
## 4 2000     2.06         0.40         0.11  5.150000  18.72727
```

When we use a which statement the vectorized code runs 20 times faster than the non-vectorized code. The which statement is much faster than the ifelse statement.