Exercise 4.7

$$\text{Runs}: \underbrace{--}_{1}\underbrace{+}_{2}\underbrace{-----}_{3}\underbrace{+}_{4}\underbrace{-}_{5}\underbrace{+}_{6}\underbrace{-}_{7}\underbrace{+++}_{8}\underbrace{--}_{9}\underbrace{++}_{10}\underbrace{--}_{11} \Rightarrow \boxed{\begin{array}{ll} R = 11 & n = 20 \\ n_1 = 8 & n_2 = 12 \end{array}}$$

$$E(R) = \frac{2n_1n_2}{n} + 1 = \frac{2 \times 8 \times 12}{20} + 1 = \boxed{10.6}$$

$$Var(R) = \frac{2n_1n_2(2n_1n_2 - n)}{n^2(n-1)} = \frac{2 \times 8 \times 12(2 \times 8 \times 12 - 20)}{20^2\, 19} = \boxed{4.345263}$$

$$z = \frac{R - E(R)}{\sqrt{Var(R)}} = \frac{11 - 10.6}{\sqrt{4.345263}} = \boxed{.1918898} \qquad \boxed{Z_{\alpha=.05} = -1.64}$$

$$\Rightarrow \quad Z < -Z_\alpha \quad (.1918 < 1.64) \Rightarrow \phi > 0 \Rightarrow \text{positive autocorrelation}$$

Predictive Analytics 1 - Homework 4

Parth Patel, Kristian Nikolov, Jieda Li, Kristiyan Dimitrov

10/19/2019

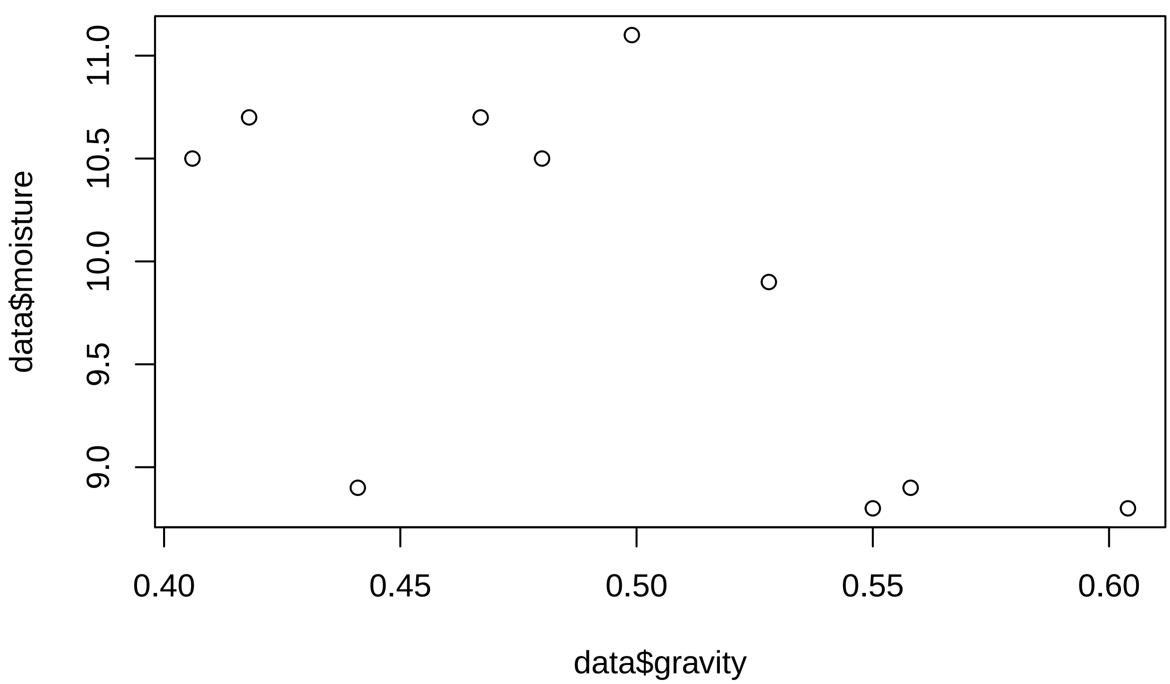
Exercise 4.8 - Woodbeam Data: Influential Observations

```
# We begin by importing the Woodbeam Data
data = read.csv("/Users/kristiyan/Documents/MSiA 401 - Predictive 1/Datasets/woodbeam.csv")
names(data) <- c("gravity","moisture","strength")
str(data)
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ gravity : num  0.499 0.558 0.604 0.441 0.55 0.528 0.418 0.48 0.406 0.467
##  $ moisture: num  11.1 8.9 8.8 8.9 8.8 9.9 10.7 10.5 10.5 10.7
##  $ strength: num  11.1 12.7 13.1 11.5 12.4 ...
```

a)

```
plot(data$gravity,data$moisture)
```



The observation with Moisture = 8.9 and spec_grav = 0.441 appears to be influential This is our 4th observation.

b)

```
X = data.matrix(data[1:2]) # Defining X matrix
X_t = t(X) # Defining X transpose matrix
product = X_t %*% X # Product of X transpose & X
product_inv = solve(product) # Calculating Inverse
```

```r
H = X %*% product_inv %*% X_t # Calculating the Hat matrix
H
```

```
##               [,1]        [,2]         [,3]       [,4]        [,5]
##  [1,] 0.151922301  0.03585072  0.006992435 0.09954523  0.03638990
##  [2,] 0.035850724  0.23648113  0.302590518 0.08255480  0.23154759
##  [3,] 0.006992435  0.30259052  0.397500716 0.08253399  0.29593616
##  [4,] 0.099545228  0.08255480  0.082533986 0.08052509  0.08159720
##  [5,] 0.036389898  0.23154759  0.295936163 0.08159720  0.22672829
##  [6,] 0.090342706  0.14109946  0.162243309 0.09022252  0.13876165
##  [7,] 0.180754473 -0.04834831 -0.111785412 0.09486478 -0.04568706
##  [8,] 0.139369819  0.04440217  0.021610233 0.09430272  0.04464126
##  [9,] 0.179655231 -0.05295295 -0.117570821 0.09301897 -0.05019916
## [10,] 0.154078997  0.01611657 -0.019624984 0.09571483  0.01711268
##              [,6]        [,7]       [,8]        [,9]       [,10]
##  [1,] 0.09034271  0.18075447 0.13936982  0.17965523  0.15407900
##  [2,] 0.14109946 -0.04834831 0.04440217 -0.05295295  0.01611657
##  [3,] 0.16224331 -0.11178541 0.02161023 -0.11757082 -0.01962498
##  [4,] 0.09022252  0.09486478 0.09430272  0.09301897  0.09571483
##  [5,] 0.13876165 -0.04568706 0.04464126 -0.05019916  0.01711268
##  [6,] 0.11664461  0.05968400 0.08892633  0.05674775  0.08099144
##  [7,] 0.05968400  0.25137759 0.16122470  0.25180391  0.19139946
##  [8,] 0.08892633  0.16122470 0.12843584  0.15999687  0.14032617
##  [9,] 0.05674775  0.25180391 0.15999687  0.25232099  0.19067037
## [10,] 0.08099144  0.19139946 0.14032617  0.19067037  0.15806344
```

```r
lev_4 = H[4,4] # Leverage of 4th observation
print(lev_4)
```

```
## [1] 0.08052509
```

Our intuition was not great. Looking at the diagonal entries, the 3rd, 5th, 7th, and 9th observations have high leverage

```r
p = 2
n = 10
print(2*(p+1)/n)
```

```
## [1] 0.6
```

```r
print(diag(H)) # These are all the leverage values from the diagonal of H (the Hat matrix)
```

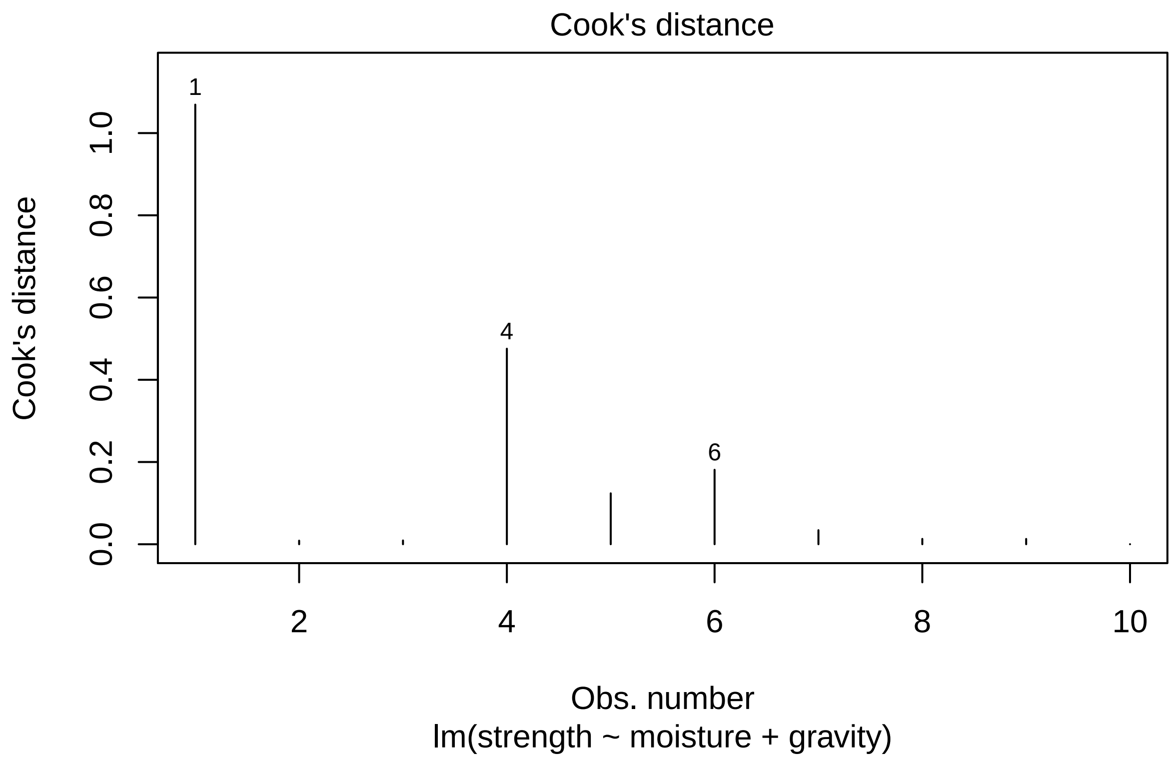
```
##  [1] 0.15192230 0.23648113 0.39750072 0.08052509 0.22672829 0.11664461
##  [7] 0.25137759 0.12843584 0.25232099 0.15806344
```

```r
print(diag(H) > (2*(p+1)/n))
```

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

It appears that based on the leverage criteria none of our observations can be considered influential.

```r
lmfit = lm(strength ~ moisture + gravity, data = data) # Fitting a line
plot(lmfit, which = 4) # Plotting Cook's Distances
```



We note that the 1st, 4th, and 6th observations are marked as influential based on Cook's Distance

Below we verify based on f-statistic

```
print(pf(.9,3,7)) # According to the documentation,
```

```
## [1] 0.5125132
```

```
# pf(probability, deg. of fr., deg. of fr.) is supposed to be the distribution function
```

```
# We use cooks.distance() function to find the numeric values of Cook's Distances
cooks.distance(lmfit)
```

```
##            1            2            3            4            5
## 1.069240e+00 8.723020e-03 9.208950e-03 4.756415e-01 1.238171e-01
##            6            7            8            9           10
## 1.810604e-01 3.418372e-02 1.303120e-02 1.288740e-02 9.905523e-05
```

```
cooks.distance(lmfit) > pf(.9,3,7)
```

```
##     1     2     3     4     5     6     7     8     9    10
##  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

We see that only the first observation is considered influential after checking both the leverage and Cook's Distances.

c)

We now fit with the first observation removed

```
new_data <- data[2:10,] # Removing the first observation, which is considered influential
str(new_data)
```

```
## 'data.frame':    9 obs. of  3 variables:
```

```
##  $ gravity : num  0.558 0.604 0.441 0.55 0.528 0.418 0.48 0.406 0.467
##  $ moisture: num  8.9 8.8 8.9 8.8 9.9 10.7 10.5 10.5 10.7
##  $ strength: num  12.7 13.1 11.5 12.4 12.6 ...
```

```r
lmfit_new = lm(strength ~ moisture + gravity, new_data)
summary(lmfit_new)
```

```
##
## Call:
## lm(formula = strength ~ moisture + gravity, data = new_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.21497 -0.05193  0.02783  0.04146  0.31137
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.59174    1.48699   5.105 0.002210 **
## moisture    -0.07314    0.09993  -0.732 0.491793
## gravity     10.26706    1.28108   8.014 0.000201 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1789 on 6 degrees of freedom
## Multiple R-squared:  0.9592, Adjusted R-squared:  0.9456
## F-statistic: 70.54 on 2 and 6 DF,  p-value: 6.79e-05
```

```r
summary(lmfit)
```

```
##
## Call:
## lm(formula = strength ~ moisture + gravity, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.44422 -0.12780  0.05365  0.10521  0.44985
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  10.3015     1.8965   5.432 0.000975 ***
## moisture     -0.2663     0.1237  -2.152 0.068394 .
## gravity       8.4947     1.7850   4.759 0.002062 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2754 on 7 degrees of freedom
## Multiple R-squared:    0.9,  Adjusted R-squared:  0.8714
## F-statistic:  31.5 on 2 and 7 DF,  p-value: 0.0003163
```

```r
# We see that all three parameters change
#       old model --> new model
# intercept: 10.3 --> 7.59;
# moisture: -0.26 --> -0.07
# gravity:  -8.49 --> 10.26
```

Exercise 4.10

a)

```r
# We import our data
data = read.csv("/Users/kristiyan/Documents/MSiA 401 - Predictive 1/Datasets/MULTDEPEND.csv")
data <- data [1:4]
str(data)
```

```
## 'data.frame':    12 obs. of  4 variables:
##  $ x1: int  8 8 8 0 0 0 2 2 2 0 ...
##  $ x2: int  1 1 1 0 0 0 7 7 7 0 ...
##  $ x3: int  1 1 1 9 9 9 0 0 0 0 ...
##  $ x4: int  1 0 0 1 1 1 1 1 1 10 ...
```

```r
print(cor(data))
```

```
##             x1          x2         x3         x4
## x1  1.00000000  0.05230658 -0.3433818 -0.4976109
## x2  0.05230658  1.00000000 -0.4315953 -0.3706964
## x3 -0.34338179 -0.43159531  1.0000000 -0.3551214
## x4 -0.49761095 -0.37069641 -0.3551214  1.0000000
```

```r
print(abs(cor(data)) < .5)
```

```
##       x1    x2    x3    x4
## x1 FALSE  TRUE  TRUE  TRUE
## x2  TRUE FALSE  TRUE  TRUE
## x3  TRUE  TRUE FALSE  TRUE
## x4  TRUE  TRUE  TRUE FALSE
```

We confirm that none of the bivariate correlations b/w x's exceed 0.5 (in absolute value)

b)

```r
library(car)
```

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

```r
data = read.csv("/Users/kristiyan/Documents/MSiA 401 - Predictive 1/Datasets/MULTDEPEND.csv")
lmfit = lm(y ~ x1 + x2 + x3 + x4, data = data )
vif(lmfit)
```

```
##       x1       x2       x3       x4
## 178.2874 158.0460 257.9074 289.3750
```

We confirm that all the VIFs are greater than 150 and the largest one is 289.375. Therefore, even though none of the bivariate correlation coefficients was very large, we still have a serious multicollinearity problem i.e. the multicolinearity is spread across all the variables.

Exercise 4.11: Gas Mileages of Cars

a)

```r
# Importing Data
data = read.csv("/Users/kristiyan/Documents/MSiA 401 - Predictive 1/Datasets/mpg.csv")
str(data)
```

```
## 'data.frame':    392 obs. of  6 variables:
```

```
##  $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : int  8 8 8 8 8 8 8 8 8 8 ...
##  $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower  : int  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : int  3504 3693 3436 3433 3449 4341 4354 4312 4425 3850 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
```

```r
cor(data[2:6]) # Calculating correlation matrix between predictor variables
```

```
##               cylinders displacement horsepower     weight acceleration
## cylinders     1.0000000    0.9508233  0.8429834  0.8975273   -0.5046834
## displacement  0.9508233    1.0000000  0.8972570  0.9329944   -0.5438005
## horsepower    0.8429834    0.8972570  1.0000000  0.8645377   -0.6891955
## weight        0.8975273    0.9329944  0.8645377  1.0000000   -0.4168392
## acceleration -0.5046834   -0.5438005 -0.6891955 -0.4168392    1.0000000
```

```r
abs(cor(data[2:6])) > .5
```

```
##              cylinders displacement horsepower weight acceleration
## cylinders         TRUE         TRUE       TRUE   TRUE         TRUE
## displacement      TRUE         TRUE       TRUE   TRUE         TRUE
## horsepower        TRUE         TRUE       TRUE   TRUE         TRUE
## weight            TRUE         TRUE       TRUE   TRUE        FALSE
## acceleration      TRUE         TRUE       TRUE  FALSE         TRUE
```

It would appear that there is high correlation (larger than .5 in absolute value) between all the predictor variables except weight & acceleration

This indicates the presence of multicollinearity.

b)

```r
lmfit = lm( mpg ~ cylinders + displacement + horsepower + weight + acceleration , data = data)
summary(lmfit) # Fitting a linear model
```

```
##
## Call:
## lm(formula = mpg ~ cylinders + displacement + horsepower + weight +
##     acceleration, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.5816  -2.8618  -0.3404   2.2438  16.3416
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.626e+01  2.669e+00  17.331   <2e-16 ***
## cylinders    -3.979e-01  4.105e-01  -0.969   0.3330
## displacement -8.313e-05  9.072e-03  -0.009   0.9927
## horsepower   -4.526e-02  1.666e-02  -2.716   0.0069 **
## weight       -5.187e-03  8.167e-04  -6.351    6e-10 ***
## acceleration -2.910e-02  1.258e-01  -0.231   0.8171
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.247 on 386 degrees of freedom
## Multiple R-squared:  0.7077, Adjusted R-squared:  0.7039
```

```
## F-statistic: 186.9 on 5 and 386 DF,  p-value: < 2.2e-16
```

Multicollinearity is reflected in the fact that three of the predictor variables are insignificant (large p values & low t-statistics) At the same time, the entire model shows a high significance via a large F-statistic = 186.9 and low p-value < 2.2e-16.

c)

```
lmfit_new = lm( mpg ~ cylinders + horsepower + weight + acceleration , data = data)
summary(lmfit_new)
```

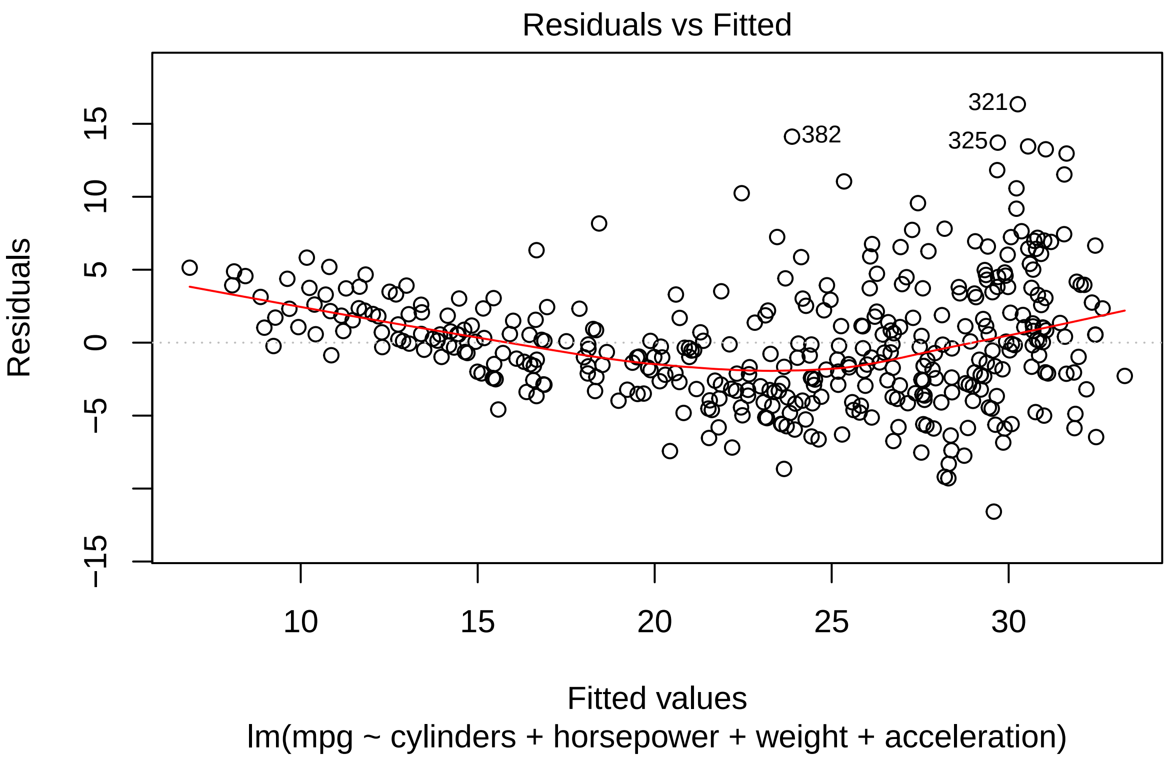
```
##
## Call:
## lm(formula = mpg ~ cylinders + horsepower + weight + acceleration,
##     data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.5807  -2.8628  -0.3409   2.2427  16.3422
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  46.2739915  2.4481591  18.902  < 2e-16 ***
## cylinders    -0.4004602  0.3032615  -1.321  0.18744
## horsepower   -0.0452970  0.0160604  -2.820  0.00504 **
## weight       -0.0051902  0.0007341  -7.070 7.26e-12 ***
## acceleration -0.0289828  0.1248944  -0.232  0.81661
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.242 on 387 degrees of freedom
## Multiple R-squared:  0.7077, Adjusted R-squared:  0.7047
## F-statistic: 234.2 on 4 and 387 DF,  p-value: < 2.2e-16
```

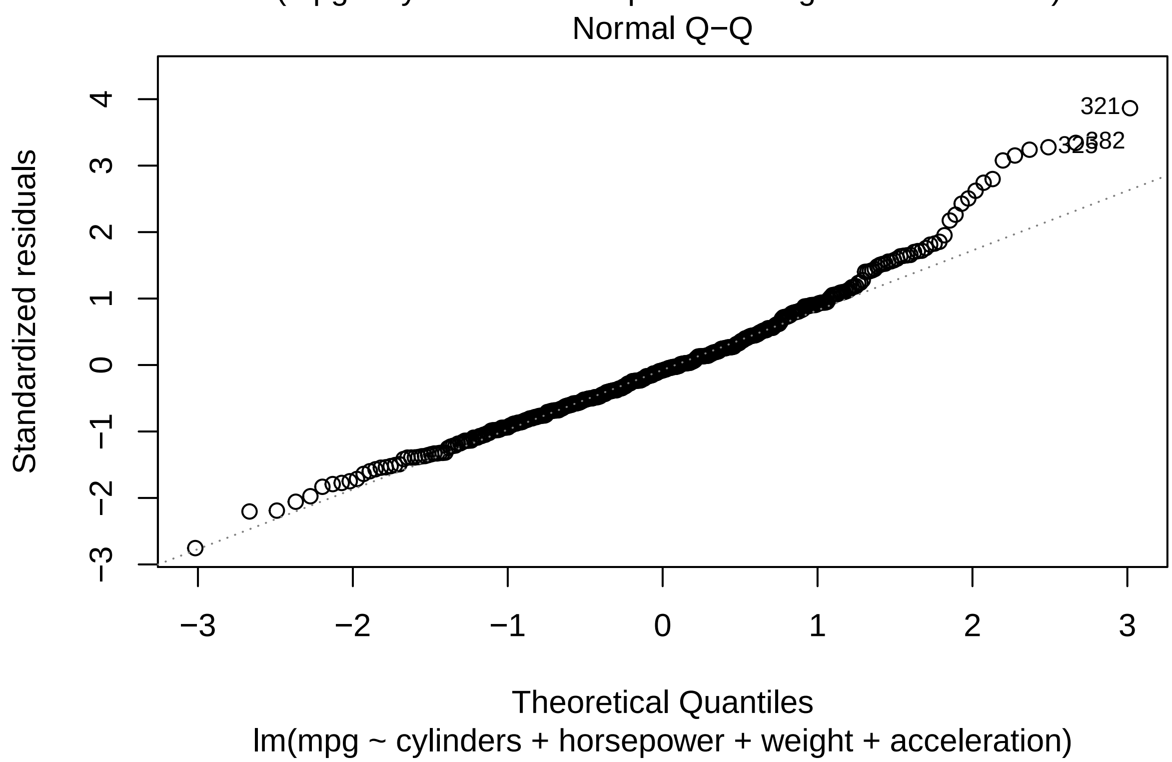
The variances for the new predictor parameters are lower than the variances of the old predictor parameters. This is reflected in the fact that the new parameters have lower p-values and higher t-statistics.

```
# p-values | old model --> new model
# -----------------------------------
#    cylinders: 0.3333 --> 0.18744;
#   horsepower: 0.0069 --> 0.00504
#       weight:  6e-10 --> 7.26e-12
# acceleration: 0.8171 --> 0.81661
```

d)

```
plot(lmfit_new, which = 1:2) # Plotting the Q-Q and residuals ~ fitted values plots
```





Indeed, we can see a roughly parabola-shape in the residuals ~ fitted values plot This suggests the inverse variance stabilizing transformation. This will help us build a model, where the homoscedasticity assumption is not violated.

```
data_transform <- data %>%
  mutate(gp100m = 100 / mpg ) %>%  # Adding the inverse column
  select(gp100m, cylinders, horsepower, weight, acceleration) # Removing mpg column
```

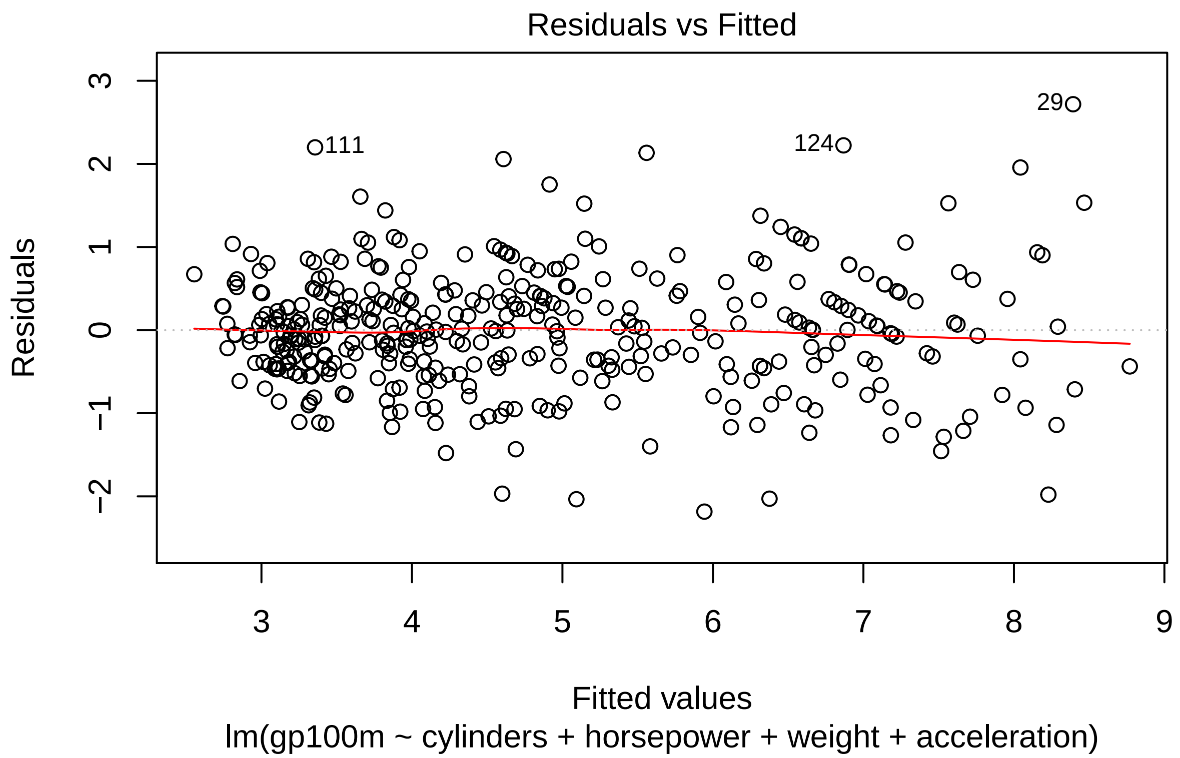
```r
str(data_transform)
```

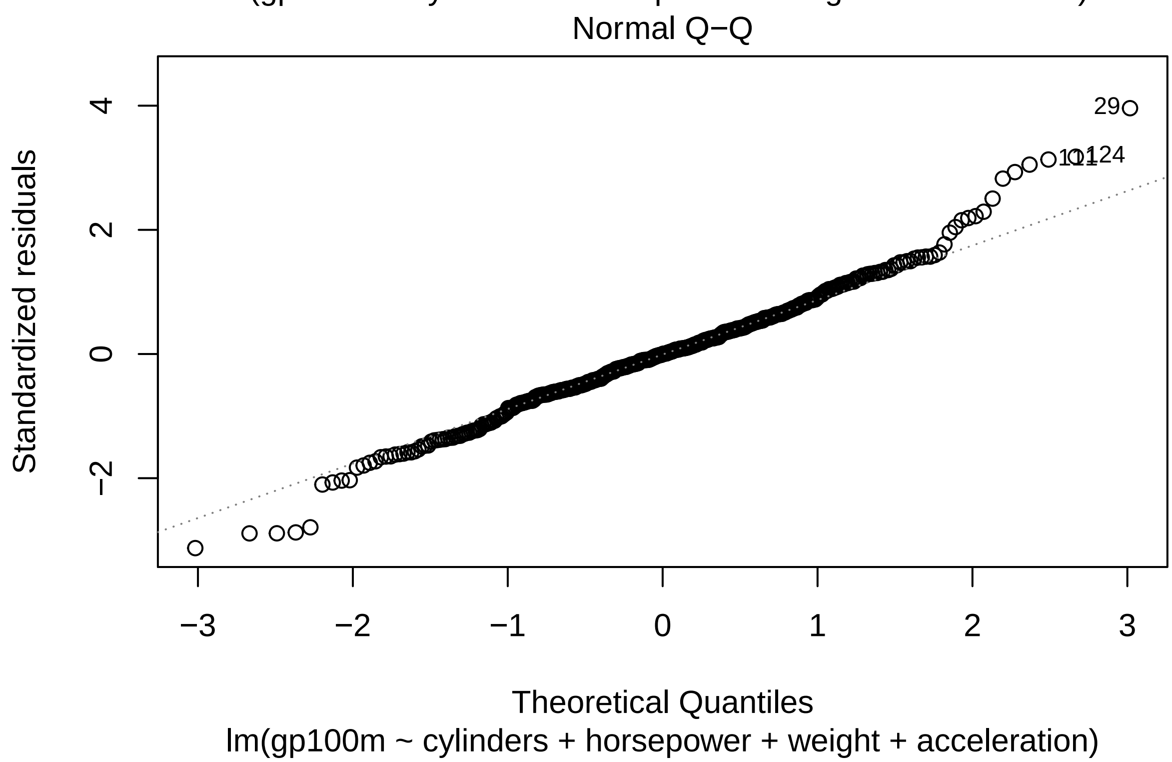
```
## 'data.frame':    392 obs. of  5 variables:
##  $ gp100m      : num  5.56 6.67 5.56 6.25 5.88 ...
##  $ cylinders   : int  8 8 8 8 8 8 8 8 8 8 ...
##  $ horsepower  : int  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : int  3504 3693 3436 3433 3449 4341 4354 4312 4425 3850 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
```

```r
# Fitting new model with the inverse transformed data
lmfit_transform = lm( gp100m ~ cylinders + horsepower + weight + acceleration, data = data_transform)
summary(lmfit_transform)
```

```
##
## Call:
## lm(formula = gp100m ~ cylinders + horsepower + weight + acceleration,
##     data = data_transform)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.18395 -0.42287 -0.00077  0.41120  2.71788
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.2174413  0.4086950  -2.979  0.00308 **
## cylinders     0.1385052  0.0506264   2.736  0.00651 **
## horsepower    0.0187233  0.0026811   6.983 1.26e-11 ***
## weight        0.0008233  0.0001225   6.718 6.59e-11 ***
## acceleration  0.0536846  0.0208498   2.575  0.01040 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7081 on 387 degrees of freedom
## Multiple R-squared:  0.8208, Adjusted R-squared:  0.8189
## F-statistic:   443 on 4 and 387 DF,  p-value: < 2.2e-16
```

```r
plot(lmfit_transform, which = 1:2) # Plotting the Q-Q and residuals ~ fitted values plots
```





We note that the variance definitely appears to be stabilized i.e. we can assume homoscedasticity under this new, transformed model The Q-Q plot, however, doesn't show any changes in the adherence to normality i.e. the tail ends continue to deviate from a normal distribution.

```
# Inspecting the VIFs
print(vif(lmfit_new)) # Model without displacement
```

```
##    cylinders   horsepower       weight acceleration
##     5.815763     8.305342     8.449468     2.580303
```

```
print(vif(lmfit_transform)) # Transformed model without displacement
```

```
##    cylinders   horsepower       weight acceleration
##     5.815763     8.305342     8.449468     2.580303
```

We conclude that the transformation has not changed the VIFs! This makes sense, the VIFs reflect correlation between the predictor variables and are therefore not influenced by the dependent variable (mpg or gp100m)

e)

```
# We now calculate a prediction interval for the next car with the specified characteristics
prediction <- predict(lmfit_transform, newdata = data.frame(cylinders = 6, horsepower = 105, weight = 30
print(paste("The estimated gallons per 100 miles for this car:",round(prediction[1],4)))
```

```
## [1] "The estimated gallons per 100 miles for this car: 4.8547"
```

```
print(paste("This, in turn, means that the miles per gallon will be:", round(100/prediction[1], 4)))
```

```
## [1] "This, in turn, means that the miles per gallon will be: 20.5985"
```

```
print(paste("The 95% prediction interval for the mpg is: [", round(100/prediction[3], 4),round(100/predi
```

```
## [1] "The 95% prediction interval for the mpg is: [ 16.0011 28.903 ]"
```

Exercise 4.12 - Acetylene data: Multicolinearity statistics

```
# Importing Data
data = read.csv("/Users/kristiyan/Documents/MSiA 401 - Predictive 1/Datasets/acetylene.csv")
str(data)
```

```
## 'data.frame':    16 obs. of  10 variables:
##  $ x1  : int  1300 1300 1300 1300 1300 1300 1200 1200 1200 1200 ...
##  $ x2  : num  7.5 9 11 13.5 17 23 5.3 7.5 11 13.5 ...
##  $ x3  : num  0.012 0.012 0.0115 0.013 0.0135 0.012 0.04 0.038 0.032 0.026 ...
##  $ x1x2: int  9750 11700 14300 17550 22100 29900 6360 9000 13200 16200 ...
##  $ x1x3: num  15.6 15.6 14.9 16.9 17.6 ...
##  $ x2x3: num  0.09 0.108 0.127 0.175 0.23 ...
##  $ x1.2: int  1690000 1690000 1690000 1690000 1690000 1690000 1440000 1440000 1440000 1440000 ...
##  $ x2.2: num  56.2 81 121 182.2 289 ...
##  $ x3.2: num  0.000144 0.000144 0.000132 0.000169 0.000182 ...
##  $ y   : num  49 50.2 50.5 48.5 47.5 44.5 28 31.5 34.5 35 ...
```

```
# Fitting the model
lmfit = lm( y ~ x1 + x2 + x3 + x1x2 + x1x3 + x2x3 + x1.2 + x2.2 + x3.2 , data = data)
summary(lmfit)
```

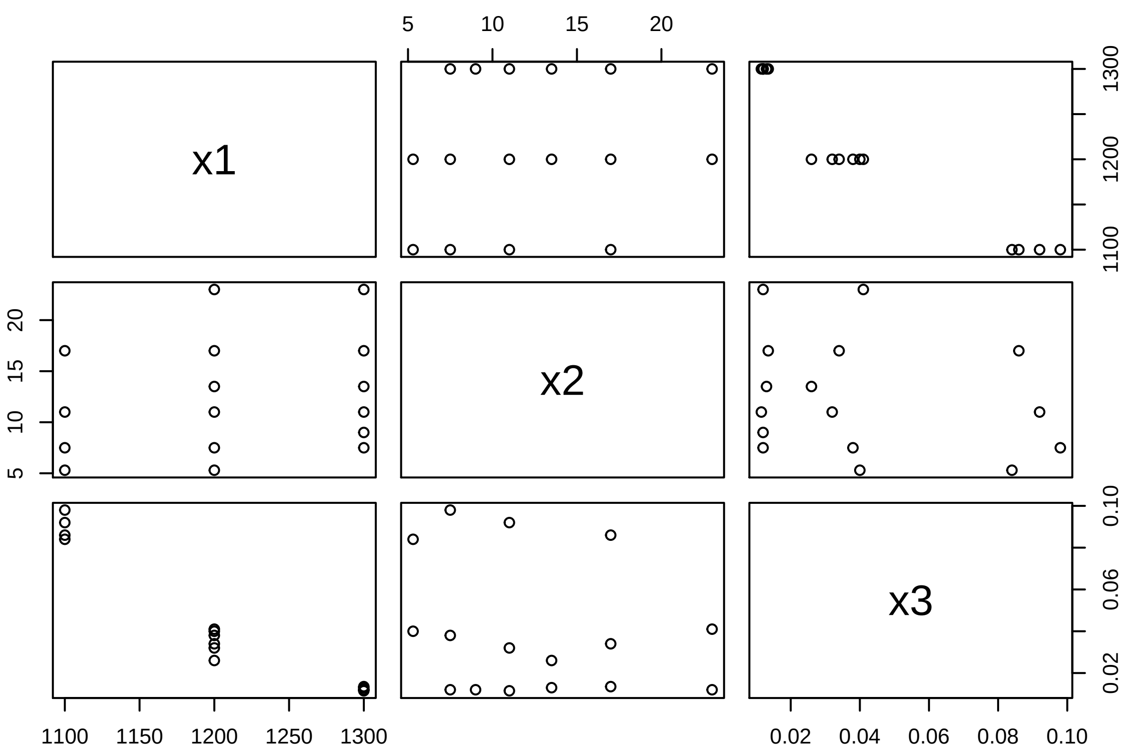
```
##
## Call:
## lm(formula = y ~ x1 + x2 + x3 + x1x2 + x1x3 + x2x3 + x1.2 + x2.2 +
##     x3.2, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.3499 -0.3411  0.1297  0.5011  0.6720
##
```

```
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.617e+03  3.136e+03  -1.153  0.29260
## x1           5.324e+00  4.879e+00   1.091  0.31706
## x2           1.924e+01  4.303e+00   4.472  0.00423 **
## x3           1.377e+04  1.045e+04   1.318  0.23572
## x1x2        -1.414e-02  3.212e-03  -4.404  0.00455 **
## x1x3        -1.058e+01  8.241e+00  -1.283  0.24666
## x2x3        -2.103e+01  9.241e+00  -2.276  0.06312 .
## x1.2        -1.927e-03  1.896e-03  -1.016  0.34874
## x2.2        -3.034e-02  1.168e-02  -2.597  0.04084 *
## x3.2        -1.158e+04  7.699e+03  -1.504  0.18318
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9014 on 6 degrees of freedom
## Multiple R-squared:  0.9977, Adjusted R-squared:  0.9943
## F-statistic: 289.7 on 9 and 6 DF,  p-value: 3.225e-07
```

Even before we dive into a), we can see signs of multicolinearity. The model as a whole appears higly significant, yet few of the variables are significant by themselves.

a)

```
plot(data[1:3]) # Plotting predictor variables against each other
```



We can see a roughly linear relationship b/w x1 & x3 (x1 goest up -> x3 goes down). This could cause multicolinearity issues. The other two variables show parallel bands i.e. no apparent relationship.

```
cor(data[1:3])# Calculating correlation matrix
```

```
##             x1          x2          x3
```

```
## x1  1.0000000  0.2236278 -0.9582041
## x2  0.2236278  1.0000000 -0.2402310
## x3 -0.9582041 -0.2402310  1.0000000
```

Here we see high correlation b/w x1 & x3, which is an indicator of multicolinearity.

b)

```r
vif(lmfit)
```

```
##           x1           x2           x3         x1x2         x1x3
## 2.856749e+06 1.095614e+04 2.017163e+06 9.802903e+03 1.428092e+06
##         x2x3         x1.2         x2.2         x3.2
## 2.403594e+02 2.501945e+06 6.573359e+01 1.266710e+04
```

Except for x2ˆ2, all other variables have very high VIFs, which indicates significant multicolinearity problems.

c)

```r
centered_data <- data %>% # Centering our data
  mutate(m_x1 = x1-mean(x1), m_x2 = x2-mean(x2), m_x3 = x3-mean(x3)) %>% # Calculating mean values of t
  mutate(m_x1x2 = m_x1*m_x2, m_x1x3 = m_x1*m_x3, m_x2x3 = m_x2*m_x3) %>% # Calculating products
  mutate(m_x1.2 = m_x1^2, m_x2.2 = m_x2^2, m_x3.2 = m_x3^2) %>% # Calculating squares
  select(y, m_x1, m_x2, m_x3, m_x1x2, m_x1x3, m_x2x3, m_x1.2, m_x2.2, m_x3.2) # Selecting only the new

str(centered_data)
```

```
## 'data.frame':    16 obs. of  10 variables:
##  $ y     : num  49 50.2 50.5 48.5 47.5 44.5 28 31.5 34.5 35 ...
##  $ m_x1  : num  87.5 87.5 87.5 87.5 87.5 87.5 -12.5 -12.5 -12.5 -12.5 ...
##  $ m_x2  : num  -4.94 -3.44 -1.44 1.06 4.56 ...
##  $ m_x3  : num  -0.0283 -0.0283 -0.0288 -0.0273 -0.0268 ...
##  $ m_x1x2: num  -432.6 -301.3 -126.3 92.4 398.7 ...
##  $ m_x1x3: num  -2.48 -2.48 -2.52 -2.39 -2.35 ...
##  $ m_x2x3: num  0.14 0.0975 0.0416 -0.0288 -0.1222 ...
##  $ m_x1.2: num  7656 7656 7656 7656 7656 ...
##  $ m_x2.2: num  24.44 11.86 2.08 1.12 20.76 ...
##  $ m_x3.2: num  0.000802 0.000802 0.00083 0.000746 0.000719 ...
```

```r
# Fitting lm model with centered data
centered_lmfit <- lm( y ~ m_x1 + m_x2 + m_x3 + m_x1x2 + m_x1x3 + m_x2x3 + m_x1.2 + m_x2.2 + m_x3.2, data
summary(centered_lmfit)
```

```
##
## Call:
## lm(formula = y ~ m_x1 + m_x2 + m_x3 + m_x1x2 + m_x1x3 + m_x2x3 +
##     m_x1.2 + m_x2.2 + m_x3.2, data = centered_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.3499 -0.3411  0.1297  0.5011  0.6720
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.590e+01  1.092e+00  32.884 5.26e-08 ***
## m_x1         4.966e-02  5.592e-02   0.888 0.408719
```

```
## m_x2         4.907e-01  5.423e-02   9.048 0.000102 ***
## m_x3        -2.542e+02  1.919e+02  -1.325 0.233461
## m_x1x2      -1.414e-02  3.212e-03  -4.404 0.004547 **
## m_x1x3      -1.058e+01  8.241e+00  -1.283 0.246663
## m_x2x3      -2.103e+01  9.241e+00  -2.276 0.063116 .
## m_x1.2      -1.927e-03  1.896e-03  -1.016 0.348741
## m_x2.2      -3.034e-02  1.168e-02  -2.597 0.040844 *
## m_x3.2      -1.158e+04  7.699e+03  -1.504 0.183182
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9014 on 6 degrees of freedom
## Multiple R-squared:  0.9977, Adjusted R-squared:  0.9943
## F-statistic: 289.7 on 9 and 6 DF,  p-value: 3.225e-07
```

```r
print(vif(centered_lmfit)) # Calculating VIFs for new centered model
```

```
##        m_x1        m_x2        m_x3      m_x1x2      m_x1x3      m_x2x3
##  375.247759    1.740631  680.280039   31.037059 6563.345193   35.611286
##      m_x1.2      m_x2.2      m_x3.2
## 1762.575365    3.164318 1156.766284
```

```r
print(vif(lmfit))
```

```
##           x1           x2           x3        x1x2        x1x3
## 2.856749e+06 1.095614e+04 2.017163e+06 9.802903e+03 1.428092e+06
##         x2x3         x1.2         x2.2         x3.2
## 2.403594e+02 2.501945e+06 6.573359e+01 1.266710e+04
```

We note that all of the VIFs are now lower with the centered data. This means that the multicollinearity problem has become less severe. Many of the VIFs, however, are still much larger than 10. Therefore, we still have a serious multicollinearity problem.

We note a significant difference between the two sets of VIFs. The VIF for the first-order term of x2 is now just 1.74, while before it was ~10^4. In the new model, only x2 & x2^2 have VIF < 10.