Multiple Regression in R

Oct., 2019 STA 206

Getting Started

In this lab of multiple regression in R, we explore a data set involving patient satisfaction. In particular, we will fit a regression model using the following as predictor variables: patient age, severity of the illness, and anxiety level (see Problem 6.15 of the textbook).

Multiple Regression in R (First-order model without Interactions)

Let's start by first reading in the data, and then renaming the columns so that we don't get the variables confused:
Read data: Before reading data, you should look at what kind of type your data file is, eg .txt; .cvs,etc, then choose the corresponding function in R such as "read.table" for txt; "read.csv" for csv.

```
  patient = read.table("patient.txt")
  names(patient) = c("satisfaction","age","severity","anxiety")
```

Exploratory data analysis:
Before we explore the relationships between variables through pairwise scatter plots, we should first examine each variable marginally: variable type, summary statistics, histogram, boxplot, pie chart, missing values?, outliers?, etc.

Check the types of variables:

```
sapply(patient, class)
satisfaction      age     severity      anxiety
"integer"    "integer"    "integer"    "numeric"
```

Notation: sapply is to apply the objective function to each element in the list.
Check number of missing values for each variable:

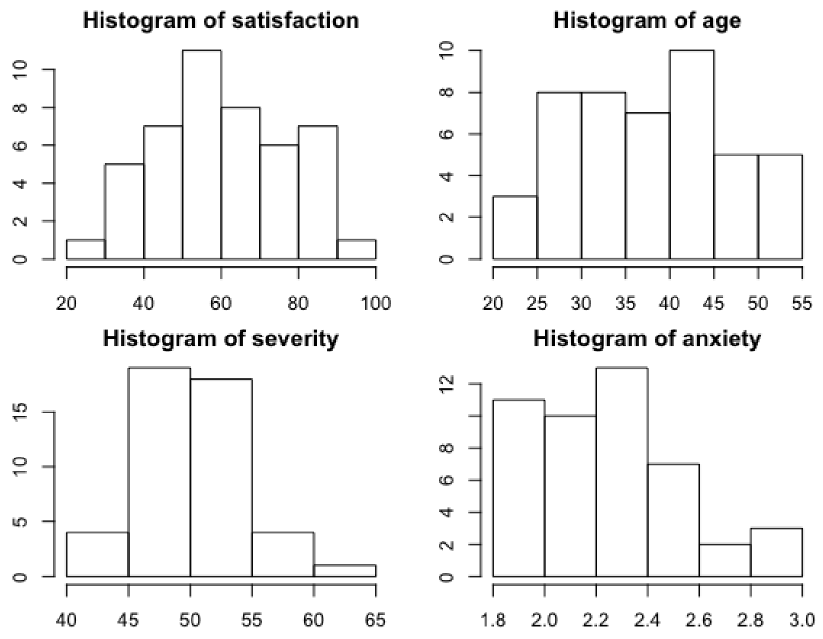
```
sapply(patient, function(x) sum(is.na(x)))
satisfaction    age     severity      anxiety
0                 0            0            0
```

Check summary statistic for each variable:

```
summary(patient)
  satisfaction        age          severity        anxiety
  Min.   :26.00   Min.   :22.00   Min.   :41.00   Min.   :1.800
  1st Qu.:48.25   1st Qu.:31.25   1st Qu.:48.00   1st Qu.:2.100
  Median :60.00   Median :37.50   Median :50.50   Median :2.300
  Mean   :61.57   Mean   :38.39   Mean   :50.43   Mean   :2.287
  3rd Qu.:76.75   3rd Qu.:44.75   3rd Qu.:53.00   3rd Qu.:2.475
  Max.   :92.00   Max.   :55.00   Max.   :62.00   Max.   :2.900
```

Plot histograms for each variable:

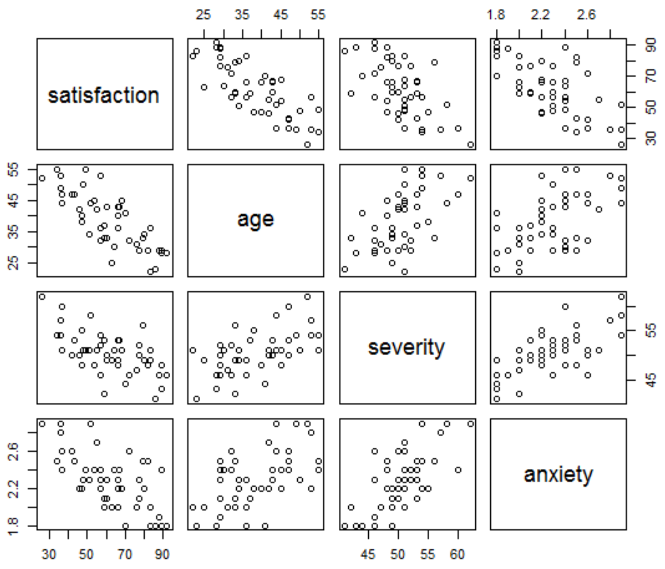
```
par(mfrow = c(2, 2))
for(i in 1:4) {
hist(patient[, i], main=paste("Histogram of", names(patient)[i]))}
```



As mentioned in previous lab, we do have some ways of seeing how the variables are related including the scatterplot matrix and the correlation matrix. They can be obtained this way:

```
  pairs(patient)  ## pairwise scatter plots
  cor(patient)    ## pairwise correlations
```

Here is the output for the first line:



and the second line gives us this matrix:

```
             satisfaction        age   severity    anxiety
satisfaction    1.0000000 -0.7867555 -0.6029417 -0.6445910
age            -0.7867555  1.0000000  0.5679505  0.5696775
severity       -0.6029417  0.5679505  1.0000000  0.6705287
anxiety        -0.6445910  0.5696775  0.6705287  1.0000000
```

Now, we will learn how to perform multiple regression in R using the lm() function. To fit the first-order model in R, we would use:

```
  fit = lm(satisfaction ~ age + severity + anxiety, data = patient)
```

Now we make use of the summary() function:

```
  summary(fit)

  Call:
  lm(formula = satisfaction ~ age + severity + anxiety, data = patient)

  Residuals:
       Min       1Q   Median       3Q      Max
  -18.3524  -6.4230   0.5196   8.3715  17.1601
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 158.4913    18.1259   8.744 5.26e-11 ***
age          -1.1416     0.2148  -5.315 3.81e-06 ***
severity     -0.4420     0.4920  -0.898   0.3741
anxiety     -13.4702     7.0997  -1.897   0.0647 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 10.06 on 42 degrees of freedom
Multiple R-squared: 0.6822,Adjusted R-squared: 0.6595
F-statistic: 30.05 on 3 and 42 DF,  p-value: 1.542e-10
```

Note that this output gives a lot of information. For instance, multiple R^2 (0.6822), adjusted R^2 (0.6595), and $\sqrt{\text{MSE}}$ (10.06). Another important piece of the output is the F-statistic, $F^* = 30.05$ and the associated degrees of freedom $(3, 42)$. Suppose we wanted to conduct the F test to determine if there is a regression relation. Here is how we could set it up ($\alpha = 0.01$):

$$H_0 : \beta_1 = \beta_2 = \beta_3 = 0 \text{ vs. } H_a : \text{ not all of the } \beta\text{'s are } 0.$$

We will reject H_0 if $F^* > F(1 - \alpha, p - 1, n - p) = F(.99, 3, 42)$. Now we can use R to find the critical value:

```
qf(.99,3,42)
[1] 4.285273
```

and from our table we know that $F^* = 30.05$. Thus, we should reject the null hypothesis that there is no regression relation. Alternatively, a p-value for this test is given by the summary, and to verify it, use:

```
1-pf(30.05,3,42)
[1] 1.543485e-10
```

Since the pvalue is less than the pre-specified significance level 0.01, we again reach the conclusion that H_0 should be rejected at level 0.01.

Another tool we have is the function confint(). With it, we can make confidence intervals for multiple parameters for any level. Suppose wanted a 97% confidence interval for the age and severity effects. We could use:

```
confint(fit,parm=c("age","severity"),level=.97)

            1.5 %     98.5 %
age      -1.62412 -0.6591038
severity -1.54712  0.6631110
```

The numbers given are the lower and upper bounds of the confidence interval, respectively. Notice how we can specify the parameters and level.

If we would like to see the ANOVA decomposition of this regression fit, use the familiar function anova():

```
anova(fit)

Analysis of Variance Table

Response: satisfaction
          Df Sum Sq Mean Sq F value    Pr(>F)
age        1 8275.4  8275.4 81.8026 2.059e-11 ***
severity   1  480.9   480.9  4.7539   0.03489 *
anxiety    1  364.2   364.2  3.5997   0.06468 .
Residuals 42 4248.8   101.2
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

To get SSE, simply look under the “Sum Sq” column for “Residuals”. To get SSR, add up the remaining elements of that same column. Finally, SSTO=SSR+SSE. The ANOVA table provides decomposition of SSR into single d.f. ESS, in the order of the predictor variables entering the model.

Source of Variation	SS	d.f.	MS
Regression	9120.5	3	3040.2
age	8275.4	1	8275.4
severity|age	480.9	1	480.9
anxiety|age,severity	364.2	1	364.2
Error	4248.8	42	101.2
Total	13369.3	45	

For example, we can obtain SSR(severity,anxiety|age) = SSR(severity|age) + SSR(anxiety|age,severity) =480.9+364.2=845.1.

Now suppose we want to get SSR(severity|age,anxiety). We need to enter the predictor variables in the following order:

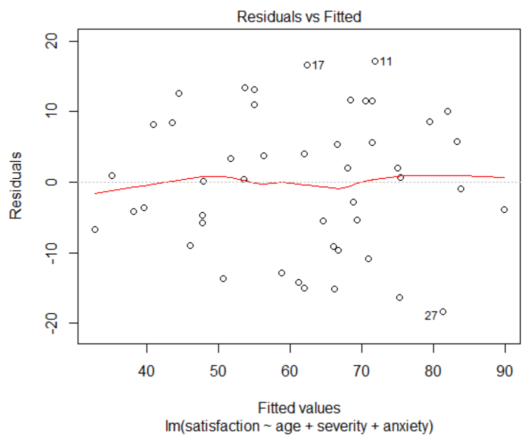
```
fit.alt = lm(satisfaction ~ age + anxiety + severity, data = patient)

anova(fit.alt)

Analysis of Variance Table

Response: satisfaction
          Df Sum Sq Mean Sq F value    Pr(>F)
age        1 8275.4  8275.4 81.8026 2.059e-11 ***
anxiety    1  763.4   763.4  7.5464  0.008819 **
severity   1   81.7    81.7  0.8072  0.374070
Residuals 42 4248.8   101.2
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

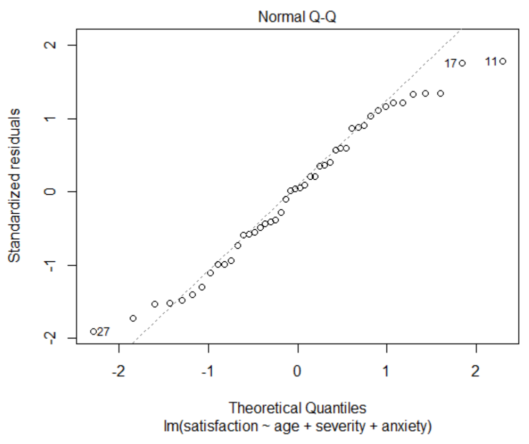
Then we can get SSR(severity|age,anxiety)=81.7.

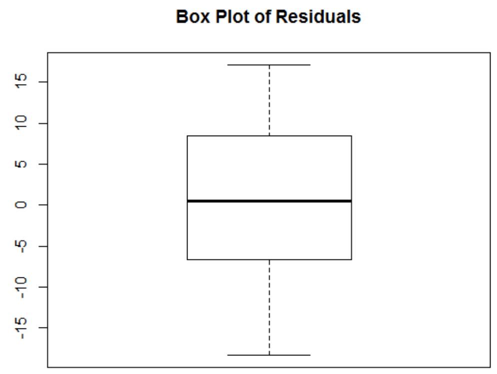
Now we might be interested in diagnostic residual plots. We can use:

```
plot(fit,which=1)  ##residuals vs. fitted values
plot(fit,which=2)  ##residuals Q-Q plot
boxplot(fit$residuals)  ## residuals boxplot
```

which give the following output:

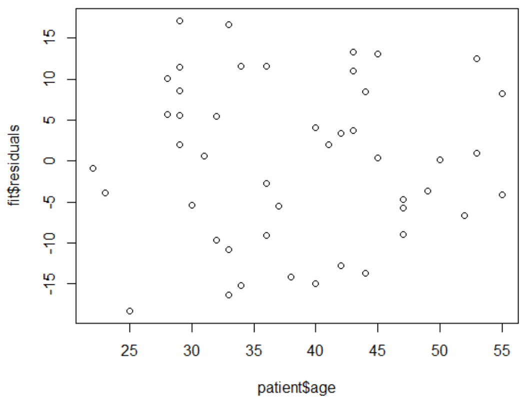


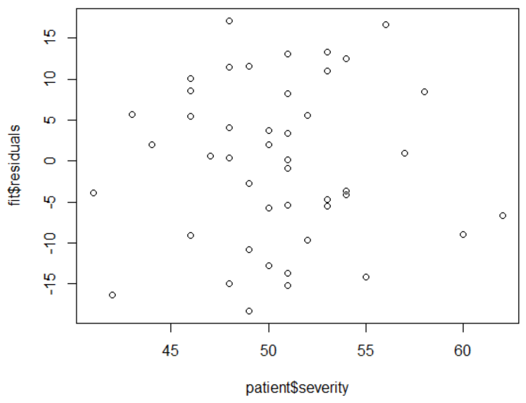


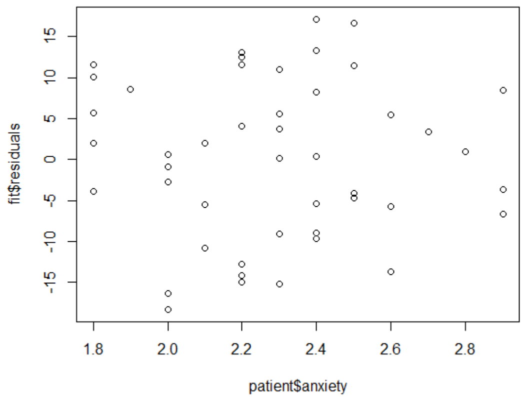


We can see that there is no obvious nonlinearity and the error distribution is a bit light-tailed.
We also want to check what the residuals versus each predictor:

```
plot(patient$age,fit$residuals)
plot(patient$severity,fit$residuals)
plot(patient$anxiety,fit$residuals)
```

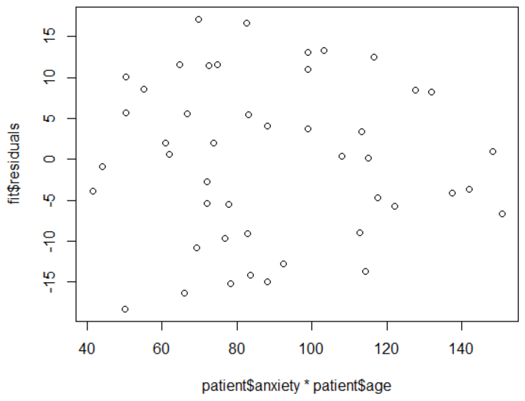


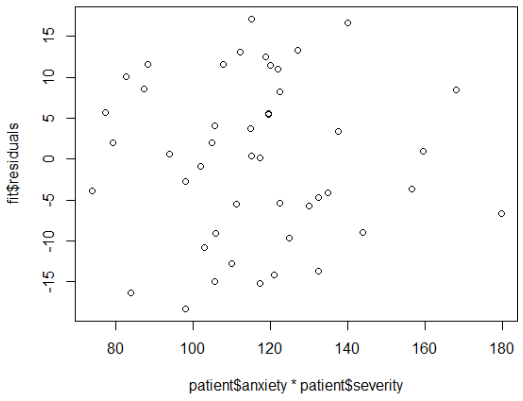


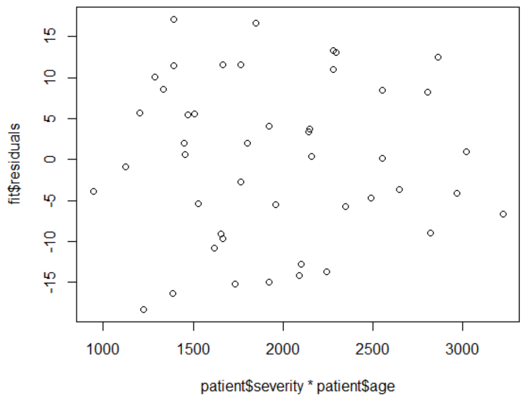


We might also want to plot the residuals against each interaction term to see if interactions should be added to our model:

```
plot(patient$anxiety*patient$age,fit$residuals)
plot(patient$anxiety*patient$severity,fit$residuals)
plot(patient$severity*patient$age,fit$residuals)
```







There is no obvious pattern in these plots, indicating that the first-order is likely to be adequate.

Multiple Regression in R (Nonadditive model with Interactions)

Now suppose we want to fit a model to this data that includes all of two-way interactions of the predictors.
The analysis can be done using the lm() function. You can add interaction terms by using a ":" in the formula. For example, if I want to include the interaction between A and B, I write *A:B*. This is what we should call for our model:
Notation: If we only want to include A,B and A:B, we can call the simplified formula A*B in lm function i.e. A+B+A:B=A*B.

```
colnames(patient)[1] = "Y"
fit2 = lm(Y ~ age+severity+anxiety+age:severity+
          age:anxiety+severity:anxiety, data=patient)
summary(fit2)

Call:
lm(formula = Y ~ age + severity + anxiety +
   age:severity + age:anxiety +
   severity:anxiety, data = patient)

Residuals:
    Min      1Q  Median      3Q     Max
-17.842  -7.151   1.279   8.052  15.063
```

```
Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      190.51810  117.37011   1.623    0.113
age                0.79293    3.15488   0.251    0.803
severity          -3.14572    3.26554  -0.963    0.341
anxiety          -14.40686   70.96754  -0.203    0.840
age:severity       0.01565    0.06396   0.245    0.808
age:anxiety       -1.19694    0.93509  -1.280    0.208
severity:anxiety   0.93330    1.54466   0.604    0.549

Residual standard error: 10.21 on 39 degrees of freedom
Multiple R-squared: 0.6957,Adjusted R-squared: 0.6489
F-statistic: 14.86 on 6 and 39 DF,  p-value: 9.358e-09
```

We can compare this summary with the summary of the first-order model, and we see that the R^2 and adjusted R^2 are relatively unchanged: R^2 is slightly increased while R_a^2 is slightly decreased. Thus, bringing in the interaction terms does not do much to account for variability in the response variable. It does not seem to be necessary to use this more complex non-additive model.
We can use the ANOVA table to get SSE, SSR, and SSTO as before:

```
anova(fit2)

Analysis of Variance Table

Response: Y
                 Df Sum Sq Mean Sq F value    Pr(>F)
age               1 8275.4  8275.4 79.3282 6.108e-11 ***
severity          1  480.9   480.9  4.6101   0.03806 *
anxiety           1  364.2   364.2  3.4908   0.06923 .
age:severity      1    8.9     8.9  0.0855   0.77148
age:anxiety       1  133.4   133.4  1.2790   0.26500
severity:anxiety  1   38.1    38.1  0.3651   0.54920
Residuals        39 4068.4   104.3
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

We notice that the SSE is relatively close to what it was before, and thus so is SSR.

SSTO should remain the same regardless of the model, since

$$\text{SSTO} = \sum_{i=1}^{n}(Y_i - \bar{Y})^2$$

does not depend on the model.

Suppose we wanted a confidence interval for the mean response at a given set of values of the X variables. For instance, let's say we want a confidence interval for the mean satisfaction when age is 19, severity is 50, and anxiety is 2.1:

```
newX = data.frame(age=19, severity=50, anxiety=2.1)
predict(fit2, newX, interval="confidence", level=0.99, se.fit=TRUE)
$fit
       fit      lwr      upr
1 83.15039 69.71023 96.59054

$se.fit
[1] 4.963288

$df
[1] 39

$residual.scale
[1] 10.21363
```

The se.fit option is telling R that we also want the standard errors. As before, we can switch the interval type to "prediction" if we want a prediction interval instead. What would happen to the size of the interval if we changed this option to "prediction"?

Finally, if you wanted to fit a model with, say, just one interaction term, you could write:

```
fit3 = lm(Y~age+severity+anxiety+anxiety:severity, data=patient)
or
fit3 = lm(Y~age+severity*anxiety, data=patient)
```

and if you wanted a model with a polynomial term, say age^2, you could write:

```
fit4 = lm(Y~age+severity+anxiety+I(age^2), data=patient)
```

If you had a model with three predictors and wanted all interactions (two-way and three-way) to be included, you could write (notice the last term):

```
  fit5 = lm(Y~X1+X2+X3+X1:X2+X1:X3+X2:X3+X1:X2:X3, data=data)
```

or simply

```
  fit5 = lm(Y~X1*X2*X3, data=data)
```

Question: If we had a model with four predictors, and we wanted to fit it using all main effects and all two-way, three-way, and four-way interactions, how many X variables are there? How many columns would the design matrix have?

Data pre-processing when dealing with missing values

In R, missing values are represented by the symbol NA (not available) . Impossible values (e.g., dividing by zero) are represented by the symbol NaN (not a number). Sometimes missing values in the original files are represented by " ", "?", "missing", etc. instead of "NA". We need to be very careful when we transform these kinds of missing values into "NA" in R.
Now we consider another data set about cars. We first read the data into R:

```
cars = read.csv('Cars.csv', header=TRUE)
```

We just focus on the variable "horsepower". We can check its class in R:

```
cars$horsepower
[1] 130 165 150 150 140 198 220 215 225 190 170 160 150 225 95  95  97
[18] 85  88  46  87  90  95  113 90  215 200 210 193 88  90  95  ?  100
...
94 Levels: ? 100 102 103 105 107 108 110 112 113 115 116 120
...

class(cars$horsepower)
[1] "factor"
```

It seems weird because we expect the class of "horsepower" to be numeric. The reason is that we have the unexpected value "?" in the data set, representing the missing value, and R will not treat this as numeric.

To transform the class of "horsepower", there is one common mistake that we may use "as.numeric" function directly:

```
as.numeric(cars$horsepower)
[1] 17 35 29 29 24 42 47 46 48 40 37 34 29 48 91 91 93 81 84 50 83 86
...
```

This will give us the indices of factor levels $(1, 2, 3, \ldots)$, rather than the actual values. For example, the first element 130 is the 17th factor levels among the 94 levels, thus its transformed value is 17.
One way we can obtain the actual values is using "as.character" and "as.numeric" functions together:

```
as.numeric(as.character(cars$horsepower))
[1] 130 165 150 150 140 198 220 215 225 190 170 160 150 225 95  95  97
[18] 85  88  46  87  90  95  113 90  215 200 210 193 88  90  95  NA  100
...
Warning message:
NAs introduced by coercion

cars$horsepower = as.numeric(as.character(cars$horsepower))
class(cars$horsepower)
[1] "numeric"
```

In fact, we can set "stringsAsFactors=FALSE" in "read.csv" function to prevent us from using the data type "factor":

```
cars = read.csv('Cars.csv', header=TRUE, stringsAsFactors=FALSE)

cars$horsepower
[1] "130" "165" "150" "150" "140" "198" "220" "215" "225" "190" "170" "160"
[13] "150" "225" "95"  "95"  "97"  "85"  "88"  "46"  "87"  "90"  "95"  "113"
[25] "90"  "215" "200" "210" "193" "88"  "90"  "95"  "?"   "100" "105" "100"
...

class(cars$horsepower)
[1] "character"

as.numeric(cars$horsepower)
[1] 130 165 150 150 140 198 220 215 225 190 170 160 150 225 95  95  97
[18] 85  88  46  87  90  95  113 90  215 200 210 193 88  90  95  NA  100
...
Warning message:
```

```
NAs introduced by coercion

cars$horsepower = as.numeric(cars$horsepower)
class(cars$horsepower)
[1] "numeric"
```