# Multiple Regression in R (Continued)

Oct., 2019 STA 206

### Getting Started

In this lab of multiple regression in R, we continue to explore the patient satisfaction data set used in the previous lab session. We will investigate the anova() output along with extra sum of squares for this data set.

### Anova and ESS

Recall the first-order model (without interactions) that we fit last week using the following code:

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

which had the following summary() output:

```
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
```

which displays many quantities of interest from our regression model ($R^2$, $R^2_{adj}$, etc.).

Another important function is anova(), which has the following output for our current model:

```
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
```

which is equivalent to the following table:

| 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.

**Q**: Can we get SSR(severity|age,anxiety) from this table?

In order 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.

Rather than using lm() and anova() function together, we can use aov() function solely to get the ANOVA table.

```
> aov(satisfaction ~ age + severity + anxiety, data = patient)
Call:
   aov(formula = satisfaction ~ age + severity + anxiety, data = patient)

Terms:
                     age severity  anxiety Residuals
Sum of Squares  8275.389  480.915  364.160  4248.841
Deg. of Freedom        1        1        1        42

Residual standard error: 10.05798
Estimated effects may be unbalanced
```

### 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.

Let’s 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"
```

### Strings in R

We will often need to manipulate strings in R e.g plotting. One of the most useful functions is *paste* which combines strings (concatentates) into one and/or multiple strings. Using the following definition, paste(..., sep=” ”, collapse=NULL), we have some quick examples.

```
> paste("x",1:3,sep='_')
```

```
[1] "x_1" "x_2" "x_3"

> paste("x",1:3,sep='_',collapse = ", ")
[1] "x_1, x_2, x_3"

> paste0("x",1:3,collapse = ',')  # paste0 is  paste with sep=""
[1] "x1,x2,x3"

> nth <- paste0(1:12, c("st", "nd", "rd", rep("th", 9)))
> nth
[1] "1st"  "2nd"  "3rd"  "4th"  "5th"  "6th"  "7th"  "8th"  "9th"
    "10th" "11th" "12th"
```

### More Data Input/Output

In addition to read.table, which deals with .txt and .csv files, we can read in excel data (.xls) through the "xlsx" package. For example,

```
> library(readxl)
> mortality = read_excel("mortality.xls")
> head(mortality)
# A tibble: 6 x 8
  PRECIP  EDUC NONWHITE  POOR   NOX   SO2 MORTALITY  CITY
    <dbl> <dbl>    <dbl> <dbl> <dbl> <dbl>     <dbl> <chr>
1     36  11.4      8.8  11.7    15    59   921.870   akr
2     35  11.0      3.5  14.4    10    39   997.875   alb
3     44   9.8      0.8  12.4     6    33   962.354   all
4     47  11.1     27.1  20.6     8    24   982.291   atl
5     43   9.6     24.4  14.3    38   206  1071.289   blt
6     53  10.2     38.5  25.5    32    72  1030.380   brm
```

We can also load html datasets directly e.g.
https://web.stanford.edu/ hastie/ElemStatLearn/datasets/bone.data.

```
> bone = read.table("https://web.stanford.edu/~hastie/
                     ElemStatLearn/datasets/bone.data", header = TRUE)
> head(bone)
idnum   age gender      spnbmd
1     1 11.70   male 0.018080670
```

```
2      1 12.70   male 0.060109290
3      1 13.75   male 0.005857545
4      2 13.25   male 0.010263930
5      2 14.30   male 0.210526300
6      2 15.30   male 0.040843210
```